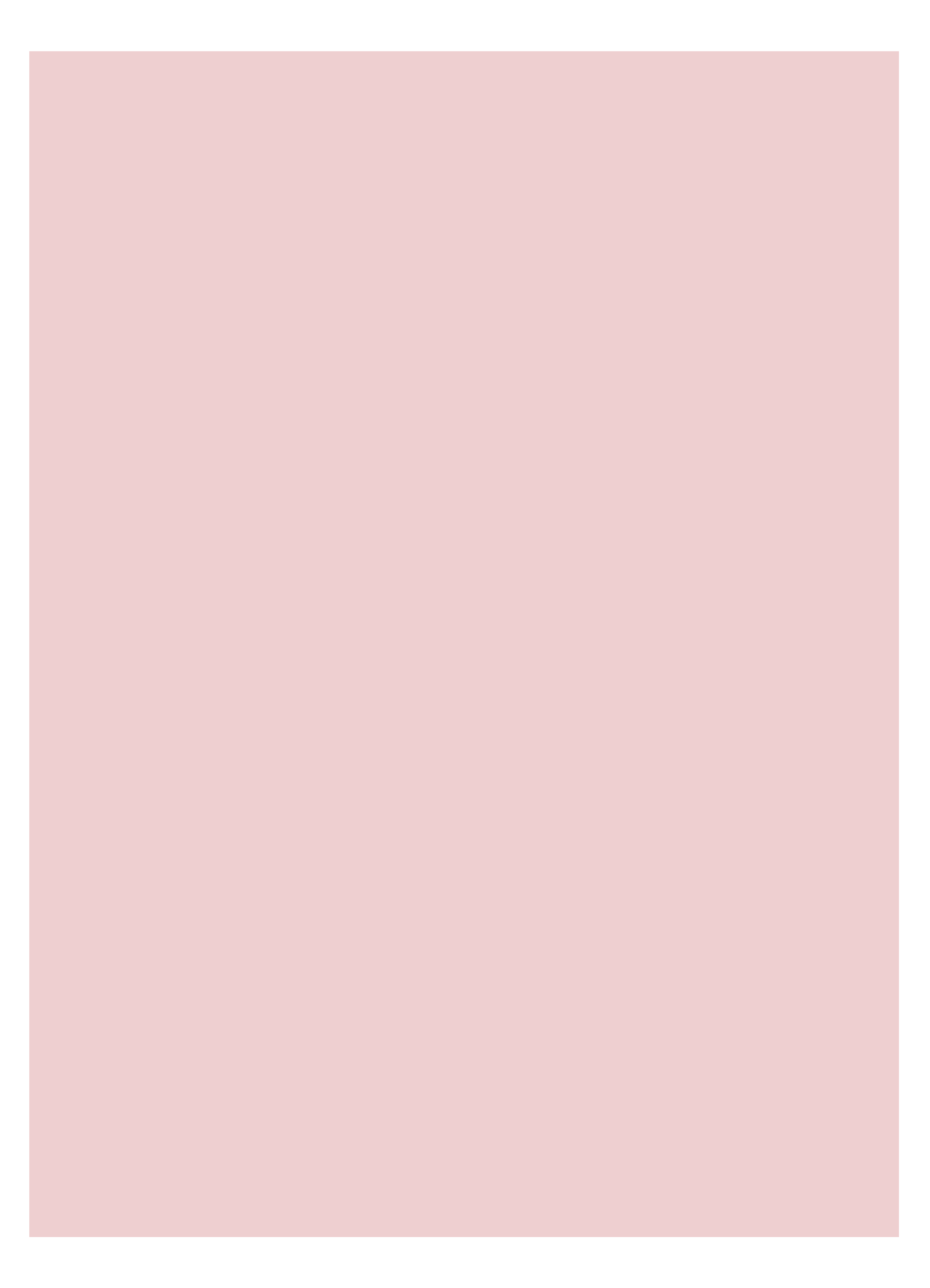
Cahiers GUTenberg

TUTORIEL TIKZ

Till Tantau

Cahiers GUTenberg, nº 48 (2007), p. 23-92.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_2007___48_23_0>





TUTORIEL TIKZ

Till Tantau

Résumé. — Karl est un prof de maths et de chimie dans un lycée. Il fabriquait auparavant les illustrations de ses textes à l'aide de l'environnement {picture} de LaTeX. Les résultats étaient certainement acceptables, mais la mise en œuvre des illustrations était toujours un travail long et pénible. Il décide d'essayer un outil dont son fils a entendu parler : Ti*k*Z. Nous suivons ses progrès au fil de son apprentissage.

Hagen doit faire un exposé sur son formalisme favori pour les systèmes distribués : les réseaux de Petri. Il découvre la puissance des outils proposés par Ti*k*Z pour tracer des ce type de structure.

Au final, nos deux cobayes semblent assez convaincus : Ti*k*Z est un logiciel de dessin technique époustouflant !

Abstract. — Karl is a math and chemistry high-school teacher. He used to create the graphics in his writings using LaTeX's {picture} environment. While the results were acceptable, creating the graphics often turned out to be a lengthy process. His son advises him to try out another tool, named Ti*k*Z. We follow him along his rapid learning curve.

Hagen must give a talk about his favorite formalism for distributed systems: Petri nets. He discovers the power of the tools available with Ti*k*Z in order to set-up this kind of structure.

At the end of the day, both of them seem rather convinced: Ti*k*Z is quite a piece of software!

Note. — Cet article est constitué de la partie intitulée « Tutorial » du manuel du logiciel PGF (`http://sourceforge.net/projects/pgf/`) de Till Tantau. Il a été traduit par Yvon Henel.

On a dû parfois faire précéder quelques uns des codes par `\shorthandoff{:!}` car PGF utilise comme délimiteurs ces caractères de ponctuaction, rendus actifs par `[frenchb]{babel}`. Sans cela une erreur est levée par une routine d'analyse lexicale. Le même problème se pose d'ailleurs avec l'utilisation de xcolor qui utilise le point d'exclamation dans la définition de couleurs. On insistera sur le fait que le passage dans une autre langue (typiquement l'anglais) *ne suffit pas* : les caractères restent actifs dans ce cas.

Cahiers GUTenberg n° 48 — Avril 2007, p. 23-92.

1. Mode d'emploi : un petit dessin pour les étudiants de Karl

Ce mode d'emploi est conçu pour les nouveaux utilisateurs de PGF et Ti*k*Z. Il ne donne pas une description exhaustive de toutes les fonctionnalités de Ti*k*Z ou de PGF mais seulement de celles dont on peut faire usage directement.

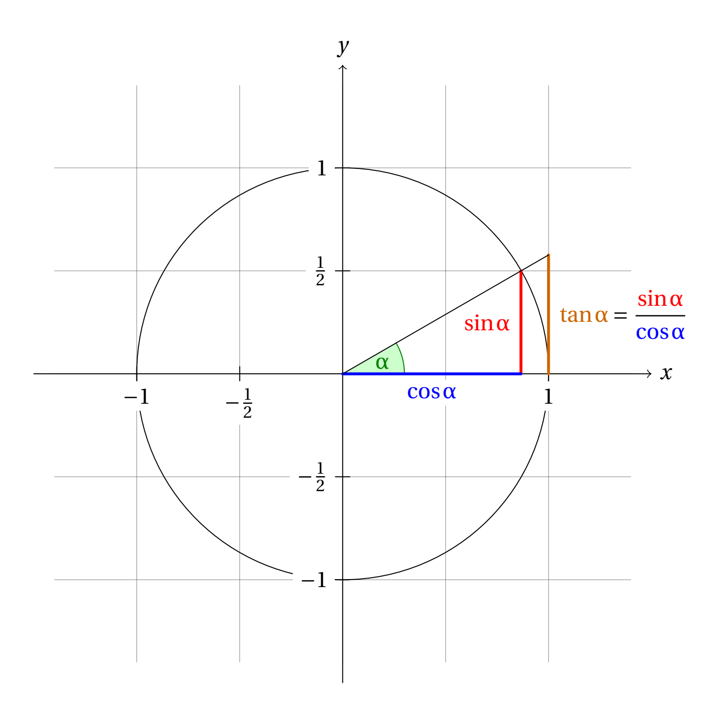
Karl est professeur de mathématiques et chimie en lycée. Il avait l'habitude de créer des graphiques dans ses feuilles d'exercices et d'examens avec l'environnement `{picture}` de LaTeX. Alors que les résultats semblaient acceptables, la création de graphiques s'avérait souvent un processus long. De plus, il y avait une tendance à avoir des problèmes avec les droites qui montraient des angles légèrement incorrects et les cercles qui semblaient être difficiles à obtenir correctement. Naturellement, ses étudiants se contrefichaient de savoir si les droites formaient les angles corrects et ils trouvaient que les examens de Karl étaient trop difficiles même si les graphiques étaient tracés avec soin. Mais Karl n'était jamais entièrement satisfait du résultat.

Le fils de Karl, qui était encore moins satisfait du résultat (il n'avait pas à passer les examens après tout), dit à Karl qu'il pouvait peut-être essayer une nouvelle extension de création de graphique. Portant un peu à confusion, cette extension semblait avoir deux noms : d'abord Karl devait télécharger et installer une extension appelée PGF. Puis il apparaissait qu'il y en avait une autre à l'intérieur, appelée Ti*k*Z, ce qui est sensé vouloir dire « Ti*k*Z ist *kein* Zeichenprogramm [1] ». Karl trouvait tout cela un peu étrange et Ti*k*Z semblait indiquer que cette extension n'était pas ce dont il avait besoin. Toutefois, ayant utilisé les logiciels GNU depuis un bon moment et « GNU n'étant pas Unix », il semblait encore y avoir de l'espoir. Son fils l'assurait que le nom de Ti*k*Z était fait pour prévenir les gens que Ti*k*Z n'est pas un logiciel que l'on peut utiliser pour faire des graphiques avec une souris ou une tablette. C'est plutôt quelque chose comme un « langage pour graphique ».

1.1. Énoncé du problème

Karl veut placer un graphique dans le prochain poly de ses étudiants. En ce moment, il enseigne le sinus et le cosinus. Ce qu'il voudrait faire est quelque chose qui ressemblerait (idéalement) à ceci :

1. Ce qui signifie Ti*k*Z n'est *pas* un programme de dessin. [N.D.T.]



L'angle α vaut 30° dans l'exemple ($\pi/6$ en radians). Le sinus de α, qui est la longueur du segment rouge est

$$\sin\alpha = 1/2.$$

Par le théorème de Pythagore, il vient $\cos^2\alpha + \sin^2\alpha = 1$. Donc la longueur du segment bleu, qui est le cosinus de α, vaut

$$\cos\alpha = \sqrt{1 - 1/4} = \tfrac{1}{2}\sqrt{3}.$$

Cela démontre que $\tan\alpha$, qui est la longueur du segment orange, vaut

$$\tan\alpha = \frac{\sin\alpha}{\cos\alpha} = 1/\sqrt{3}.$$

1.2. Préparer l'environnement

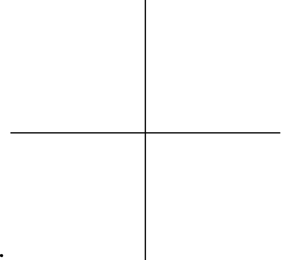
Pour dessiner une figure avec TikZ on doit au début de celle-ci dire à TeX ou LaTeX que l'on veut commencer une figure. En LaTeX on utilisera l'environnement `{tikzpicture}`, en « plain TeX » on commencera la figure par `\tikzpicture` et on la finira par `\endtikzpicture`.

1.2.1. *Préparation de l'environnement en LaTeX*

Karl, qui utilise LaTeX, créera donc son fichier comme suit :

```
\documentclass{article} %par exemple
\usepackage{tikz}
\begin{document}
Nous travaillons sur
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0); \draw (0,-1.5) -- (0,1.5);
\end{tikzpicture}.
\end{document}
```

Une fois le fichier exécuté, c.-à-d. compilé par `pdflatex`, ou par `latex` suivi de `dvips`, le résultat contiendra quelque chose qui ressemblera à ceci :

Nous travaillons sur  .

D'accord, pas encore toute la figure pour l'instant, mais nous avons déjà placé les axes. Enfin, pas tout à fait, mais nous avons des droites qui deviendront des axes. Soudain, Karl a le sentiment poignant que la figure est encore loin.

Regardons ce code de plus prêt. D'abord, on charge l'extension TIKZ. Cette extension est appelée « interface » au système de base de PGF. La couche de base, décrite dans le manuel complet, est quelque peu, disons, basique et, de ce fait, plus difficile à utiliser. L'interface rend les choses plus faciles en fournissant une syntaxe plus simple.

Dans l'environnement il y a deux commandes `\draw`. Elles signifient : « Le chemin, qui est spécifié après la commande et jusqu'au point-virgule, doit être tracé. » Le premier chemin est déterminé par `(-1.5,0) -- (0,1.5)` ce qui signifie « un segment de droite depuis le point à la position $(-1.5,0)$ jusqu'au point à la position $(0,1.5)$ ». Ici, les positions sont déterminées dans un repère spécial pour lequel une unité, au départ, vaut 1 cm.

Karl est assez content de voir que l'environnement réserve automatiquement assez d'espace pour contenir toute la figure.

1.2.2. *Préparation de l'environnement en Plain TEX*

Il se trouve que Gerda, la femme de Karl, qui est aussi professeur de mathématiques, n'utilise pas LATEX mais plain TEX car elle préfère faire les choses à « l'ancienne ». Elle aussi peut utiliser Ti*k*Z. Au lieu de `\usepackage{tikz}` elle doit écrire `\input tikz.tex`, au lieu de `\begin{tikzpicture}` elle écrit `\tikzpicture` et au lieu de `\end{tikzpicture}` elle écrit `\endtikzpicture`.

Elle utiliserait donc :

```
%% fichier Plain TeX
 \input tikz.tex
 \baselineskip=12pt \hsize=6.3truein \vsize=8.7truein
 Nous travaillons sur
 \tikzpicture
 \draw (-1.5,0) -- (1.5,0); \draw (0,-1.5) -- (0,1.5);
 \endtikzpicture.
 \bye
```

Gerda peut compiler ce fichier avec `pdftex`, ou bien `tex` suivi de `dvips`. Ti*k*Z détectera automatiquement le pilote utilisé. Si elle désire utiliser `dvipdfm` avec `tex`, elle devra soit modifier le fichier `pdf.cfg` ou écrire `\def\pgfsysdriver{pgfsys-dvipdfm.def}` quelque part *avant* qu'elle ne charge `tikz.tex` ou `pgf.tex`.

1.2.3. *Préparation de l'environnement en ConTEXt*

Hans, l'oncle de Karl, utilise ConTEXt. Comme Gerda, Hans peut aussi utiliser Ti*k*Z. Au lieu de `\usepackage{tikz}`, il écrit `\usemodule[tikz]`. Au lieu de `\begin{tikzpicture}` il écrit `\starttikzpicture` et à la place de `\end{tikzpicture}` il écrit `\stoptikzpicture`.

Sa version de l'exemple est la suivante :

```
%% fichier ConTeXt
\usemodule[tikz]

\starttext
  Nous travaillons sur
  \starttikzpicture
    \draw (-1.5,0) -- (1.5,0);
    \draw (0,-1.5) -- (0,1.5);
  \stoptikzpicture.
\stoptext
```

Hans compilera maintenant son fichier comme d'habitude avec `texexec`.

1.3. Construction d'un chemin droit

Le bloc de base de toute figure dans Ti*k*Z est le chemin (*path*). Un chemin est une suite de segments de droites ou de courbes qui sont connectés (ce n'est pas le tout de la chose mais laissons les complications pour l'instant). On commence un chemin en donnant les coordonnées du point représentant sa position de départ, entre parenthèses, comme

dans (0,0). On continue par une suite « d'opérations d'extension de chemin ». La plus simple est -- que l'on a déjà utilisée. Elle doit être suivie d'autres coordonnées et étend le chemin en ligne droite jusqu'à la nouvelle position. Par exemple, si nous transformions les deux chemins des axes en un seul, ceci s'en suivrait :

```
\tikz \draw (-1.5,0) -- (1.5,0)
            -- (0,-1.5) -- (0,1.5);
```

Karl est un peu troublé par le fait qu'il n'y a pas d'environnement {tikzpicture} ici. À la place, on utilise la commande \tikz. Cette commande prend soit un seul argument (commençant par une accolade ouvrante comme dans \tikz{\draw (0,0) -- (1.5,0)} dont résulte ———), soit absorbe tout ce qui se trouve avant le point-virgule suivant et le place dans un environnement {tikzpicture}. En gros, toutes les commandes de dessin de Ti*k*Z doivent apparaître comme argument de \tikz ou dans un environnement {tikzpicture}. Heureusement, la commande \draw n'est définie que dans cet environnement, il y a donc peu de risques pour que l'on fasse, accidentellement, quelque chose de travers.

1.4. Construction de chemin courbe

Ce que veut faire Karl ensuite c'est tracer un cercle. Pour ça, évidemment, on ne s'en sortira pas avec des droites. Nous avons besoin, au contraire, de tracer des courbes. Pour cela, Ti*k*Z fournit une syntaxe spéciale. On a besoin de un ou deux « points de contrôle ». Les maths cachées derrière ne sont pas tout à fait triviales mais voici l'idée de base : supposez que l'on soit au point x et que le premier point de contrôle soit y. Alors la courbe va démarrer « dans la direction de y en partant de x », c'est-à-dire que la tangente à la courbe en x va pointer vers y. Ensuite, supposons que la courbe doive finir en z avec w pour deuxième point de contrôle. Alors la courbe finira bien en z et la tangente à la courbe au point z passera par w.

Voici un exemple (on a ajouté les points de contrôle pour plus de clarté) :

```
\begin{tikzpicture}
  \filldraw [gray] (0,0) circle (2pt)
                   (1,1) circle (2pt)
                   (2,1) circle (2pt)
                   (2,0) circle (2pt);
  \draw (0,0) .. controls (1,1) and (2,1) .. (2,0);
\end{tikzpicture}
```

La syntaxe générale pour étendre un chemin d'une manière « courbe » est `.. controls` *premier point de contrôle* `and` *second point de contrôle* `..` *point final*. On peut ne pas utiliser le `and` *second point de contrôle* ce qui fera que le premier sera utilisé deux fois.

Désormais, Karl peut ajouter le premier demi-cercle à la figure :

```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (-1,0) .. controls (-1,0.555)
                  and (-0.555,1) .. (0,1)
               .. controls (0.555,1)
                  and (1,0.555) .. (1,0);
\end{tikzpicture}
```

Karl est très content du résultat mais trouve que décrire des cercles de cette manière est extrêmement maladroit. Heureusement, il y a une manière bien plus simple.

1.5. Construction d'un chemin circulaire

Afin de tracer un cercle, on peut utiliser l'opération de construction de chemin `circle` (*cercle*). Cette opération est suivie du rayon entre parenthèses comme dans l'exemple suivant (Notez que la position précédente est utilisée comme *centre* du cercle.) :

```
\tikz \draw (0,0) circle (10pt);
```

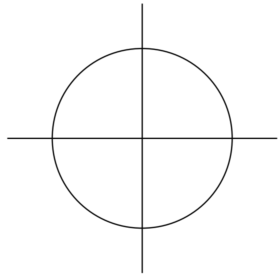
On peut également ajouter une ellipse au chemin avec l'opération `ellipse`. Au lieu d'un seul rayon on en donne deux, l'un pour la direction des x et l'autre pour celle des y, séparés par `and` :

```
\tikz \draw (0,0) ellipse (20pt and 10pt);
```

Pour tracer une ellipse dont les axes ne sont pas horizontaux et verticaux mais pointent dans une direction arbitraire (une « ellipse tournée » comme ⬯) on peut utiliser des transformations, ce que l'on expliquera plus loin. En passant, le code de la petite ellipse est `\tikz \draw[rotate=30] (0,0) ellipse (6pt and 3pt);`.

Donc, pour retourner au problème de Karl, il peut écrire `\draw (0,0) circle (1cm);` pour tracer un cercle.

```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
\end{tikzpicture}
```

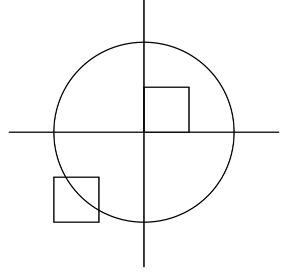


À ce moment-là, Karl est un peu alarmé de voir ce cercle si petit quand il voudrait que la figure finale soit bien plus grande. Il est content d'apprendre que Ti*k*Z possède de puissantes options de transformation et qu'agrandir tout d'un facteur quelconque est très facile. Mais gardons la taille comme elle est pour le moment afin de gagner un peu de place.

1.6. Construction d'un chemin rectangulaire

La prochaine chose que nous voudrions avoir est un quadrillage de fond. Il y a plusieurs manières de l'obtenir. Par exemple, on peut tracer un grand nombre de rectangles. Comme les rectangles sont si courants, il y a une syntaxe spéciale pour eux : pour ajouter un rectangle au chemin courant, utiliser l'opération de construction de chemin `rectangle`. Cette opération doit être suivie d'autres coordonnées et ajoutera un rectangle au chemin de telle sorte que les coordonnées précédentes et les suivantes déterminent les coins du rectangle. Ajoutons donc deux rectangles à la figure :

```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
  \draw (0,0) rectangle (0.5,0.5);
  \draw (-0.5,-0.5) rectangle (-1,-1);
\end{tikzpicture}
```



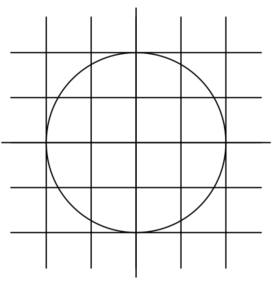
Bien que ça puisse être sympa dans d'autres circonstances, ça ne nous aide pas vraiment pour le problème de Karl : d'abord il nous faudrait bien trop de ces rectangles et puis la bordure n'est pas « fermée ».

Aussi, Karl est prêt à recourir simplement au tracé de quatre verticales et quatre horizontales en utilisant l'aimable commande `\draw` lorsqu'il apprend qu'il y a aussi une opération de construction de chemin `grid` (*quadrillage*).

1.7. Construction d'un quadrillage

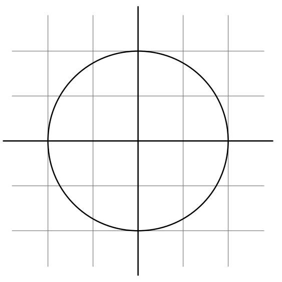
Karl pourrait utiliser le code suivant :

```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
  \draw[step=.5cm] (-1.4,-1.4) grid (1.4,1.4);
\end{tikzpicture}
```



Après un autre coup d'œil sur la figure attendue, Karl remarque que ça serait mieux si le quadrillage était moins visible. (Son fils lui a dit que les quadrillages ont tendance à troubler le lecteur s'ils sont trop visibles.) Pour cela, Karl ajoute deux nouvelles options à la commande `\draw` qui dessine le quadrillage. D'abord il utilise la couleur `gray` pour les lignes du quadrillage. Ensuite il réduit la largeur des lignes avec `very thin`. Enfin il échange les places des commandes afin que le quadrillage soit tracé d'abord et que tout le reste vienne par dessus.

```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin]
              (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
\end{tikzpicture}
```



1.8. Ajouter une touche de style

Au lieu d'utiliser les options `gray, very thin`, Karl aurait pu tout aussi bien écrire `style=help lines`. Les *styles* sont des ensembles prédéfinis d'options que l'on peut utiliser pour organiser la manière dont

le graphique sera tracé. En écrivant `style=help lines` on dit : « utilise le style que moi (ou un autre) j'ai défini pour tracer les *lignes d'aide* ». Si Karl décide, plus tard, que les quadrillages devraient être tracés avec, par exemple, la couleur `blue!50` au lieu de `gray`, il lui suffirait d'écrire ce qui suit :

```
\tikzstyle help lines=[color=blue!50,very thin]
```

Ou il pourrait écrire :

```
\tikzstyle help lines+=[color=blue!50]
```

Cela ajouterait l'option `color=blue!50`. Le style `help lines` contiendrait maintenant *deux* options de couleur mais la deuxième remplacerait la première.

En utilisant les styles on rend le code des graphiques plus flexible. On peut changer l'aspect des choses d'une manière plus cohérente.

On peut construire une structure hiérarchique de styles en définissant des styles à l'aide d'autres. Ainsi, afin de définir un style `Karl's grid` basé sur le style `grid`, Karl pourrait écrire

```
\tikzstyle Karl's grid=[style=help lines,color=blue!50]
...
\draw[style=Karl's grid] (0,0) grid (5,5);
```

On peut ne pas écrire le `style=`. En fait, à chaque fois que Ti*k*Z rencontre une option qu'il ne connaît pas, il vérifie si cette option n'est pas un nom de style. Si oui, le style est utilisé. Ainsi, Karl pourrait avoir écrit :

```
\tikzstyle Karl's grid=[help lines,color=blue!50]
...
\draw[Karl's grid] (0,0) grid (5,5);
```

Pour certains styles, comme `very thin`, on voit clairement ce que le style fait et il n'est pas nécessaire d'écrire `style=very thin`. Pour d'autres, comme `help lines`, il me semble qu'écrire `style=help lines` est plus naturel. Mais c'est essentiellement une question de goût.

1.9. Option de dessin

Karl se demande quelles sont les autres options qui influent sur la façon dont un chemin est tracé. Il a déjà vu que l'on peut utiliser l'option `color=`*color* pour fixer la couleur d'une ligne. L'option `draw=`*color* fait presque la même chose, seulement elle fixe uniquement la couleur des traits et on peut utiliser une autre couleur pour le remplissage (Karl en aura besoin pour colorier le secteur angulaire).

Il remarque que le style `very thin` produit des traits très fins. Karl n'en est pas vraiment surpris ni par le fait que `thin` produise des traits fins, `thick` des traits épais, `very thick` des traits très épais, `ultra thick` des traits vraiment très, très épais et que `ultra thin` produise des traits si fins que certaines imprimantes basse définition ou certains écrans ont du mal à les montrer. Il se demande quelle option permet d'obtenir des traits d'épaisseur « normale ». Il se trouve que c'est `thin` qui le fait. Karl trouve cela étrange mais son fils lui explique que LaTeX a deux commandes nommées `\thinlines` et `\thicklines` et que `\thinlines` donne les traits d'épaisseur « normale », plus précisément, de l'épaisseur, par exemple, du fût d'une lettre comme « T » ou « i ». Néanmoins, Karl voudrait savoir s'il existe quelque chose entre `thin` et `thick`. Il y a quelque chose, c'est `semithick`.

Une autre chose que l'on peut faire c'est tracer les lignes en pointillés ou avec des tirets. Pour cela on peut utiliser les styles `dashed` et `dotted` ce qui produit - - et ...... Ces deux options existent dans une version lâche (*loose*) et une version serrée (*dense*) appelée `loosely dashed`, `densely dashed`, `loosely dotted` et `densely dotted`. S'il y tient vraiment beaucoup Karl peut aussi définir des motifs plus complexes de pointillés avec l'option `dash pattern` mais son fils insiste sur le fait que les pointillés doivent être utilisés avec la plus grande prudence et que, en général, ils perturbent le lecteur. Le fils de Karl maintient que les motifs compliqués de pointillés sont mauvais. Les étudiants de Karl se contrefichent des motifs de pointillés.

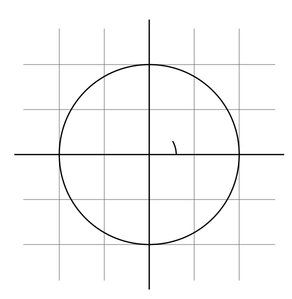
1.10. Construction d'arc

Notre prochain obstacle est de dessiner un arc pour marquer l'angle. Pour cela, l'opération de construction de chemin `arc` se révèle utile. Elle trace une partie de cercle ou d'ellipse. Cette opération `arc` doit être suivie par un triplet entre parenthèses. Les deux premières composantes sont des angles, la dernière est un rayon. Par exemple, `arc(10:80:10pt)` signifie « un arc allant de 10 à 80 degrés sur un cercle de rayon 10pt ». Karl a évidemment besoin d'un arc de 0° à 30°. Le rayon devrait être assez petit, peut-être environ un tiers du rayon du cercle. Cela donne `(0:30:3mm)`.

Lorsque l'on utilise l'opération de construction de chemin `arc`, l'arc déterminé est ajouté avec pour point de départ la position courante.

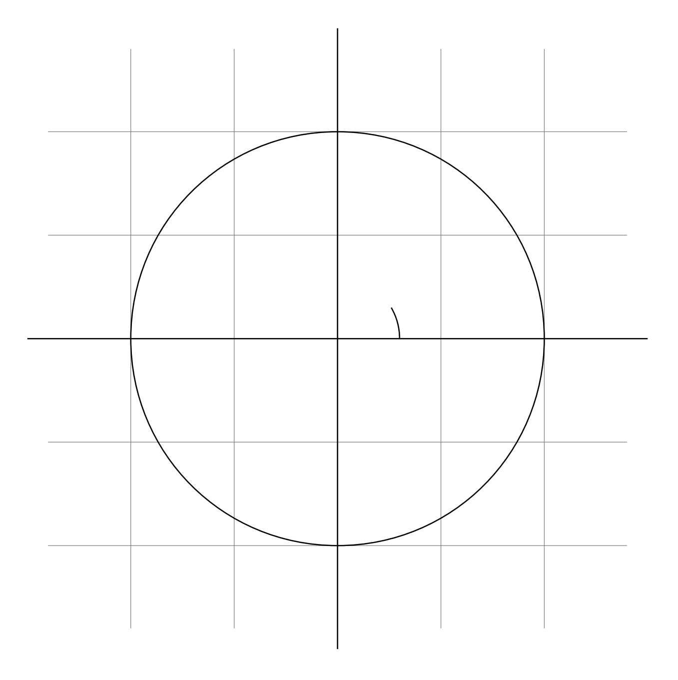
Nous avons donc besoin d'abord d'y aller.

```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin]
            (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
  \draw (3mm,0mm) arc (0:30:3mm);
\end{tikzpicture}
```



Karl pense que tout cela est un peu trop petit et qu'il ne peut pas continuer sans apprendre comment mettre à l'échelle. Pour cela, il peut ajouter l'option `[scale=2.3]`. Il peut ajouter cette option à chaque commande `\draw` mais ce serait maladroit. Au lieu de cela il l'ajoute à tout l'environnement, ce qui conduit cette option à s'appliquer partout à l'intérieur.

```
\begin{tikzpicture}[scale=2.3]
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
  \draw (3mm,0mm) arc (0:30:3mm);
\end{tikzpicture}
```



Comme pour les cercles, on peut spécifier « deux » rayons pour obtenir un arc d'ellipse.

```
\tikz \draw (0,0) arc
            (0:315:1.75cm and 1cm);
```

1.11. Découpage d'un chemin

Pour gagner un peu de place dans cet article il serait bien de rogner un peu le graphique de Karl afin de pouvoir nous concentrer sur les parties « intéressantes ». Découper est vraiment simple avec Ti*k*Z. On utilise la commende `\clip` qui découpe tout le dessin qui suit. Cela marche comme `\draw` mais au lieu de tracer quelque chose ça utilise le chemin donné pour découper tout ce qui suit.

```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin]
            (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
  \draw (3mm,0mm) arc (0:30:3mm);
\end{tikzpicture}
```

On peut faire les deux choses en même temps : dessiner *et* découper un chemin. Pour cela, on utilise la commande `\draw` en ajoutant l'option `clip`. (Ce n'est pas tout : on peut également utiliser la commande `\clip` avec l'option `draw`. Bon, ce n'est pas tout non plus : en fait, `\draw` est juste une abréviation pour `\path[draw]` et `\clip` une abréviation pour `\path[clip]` et on peut aussi écrire `\path[draw,clip]`.) Voici un exemple :

```
\begin{tikzpicture}[scale=3]
  \clip[draw] (0.5,0.5) circle (.6cm);
  \draw[step=.5cm,gray,very thin]
          (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
  \draw (3mm,0mm) arc (0:30:3mm);
\end{tikzpicture}
```

1.12. Construction de chemin parabolique ou sinusoïdal

Bien que Karl n'en ait pas besoin pour son graphique, il est content d'apprendre qu'il y a des opérations de chemin `parabola` (*parabole*), `sin` et `cos` pour ajouter des courbes paraboliques ou sinusoïdales au chemin courant. Pour l'opération `parabola` le point courant sera sur la parabole ainsi que le point donné après l'opération. Considérons l'exemple suivant :

```
\tikz \draw (0,0) rectangle (1,1)
                    (0,0) parabola (1,1);
```

On peut aussi placer le sommet ailleurs :

```
\tikz \draw[x=1pt,y=1pt] (0,0) parabola bend (4,16) (6,12);
```

Les opérations `sin` et `cos` ajoutent une courbe d'équation $y = \sin x$ ou $y = \cos x$ pour x dans l'intervalle $[0, \pi/2]$ de telle sorte que le point courant soit le point de départ de la courbe et que celle-ci finisse au point donné. Voici deux exemples :

```
Une sinusoïde \tikz \draw[x=1ex,y=1ex]
                    (0,0) sin (1.57,1);.
```

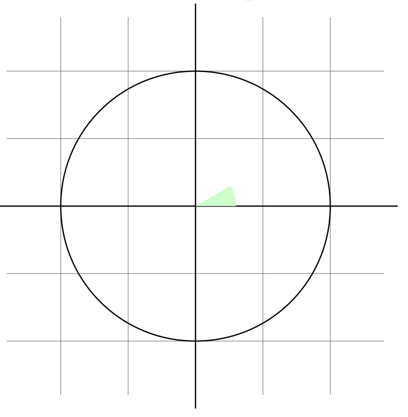
Une sinusoïde.

```
\tikz \draw[x=1.57ex,y=1ex]
    (0,0) sin (1,1) cos (2,0) sin (3,-1) cos (4,0)
    (0,1) cos (1,0) sin (2,-1) cos (3,0) sin (4,1);
```

1.13. Tracer et colorier

Revenant à la figure, Karl veut maintenant colorier l'angle avec un vert très pâle. Pour cela il utilise `\fill` au lieu de `\draw`. Voici ce que fait Karl :

```
\begin{tikzpicture}[scale=1.5]
  \draw[step=.5cm,gray,very thin]
       (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
\fill[green!20!white] (0,0) --
  (3mm,0mm) arc (0:30:3mm) -- (0,0);
\end{tikzpicture}
```



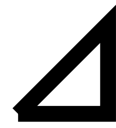
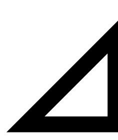
La couleur `green!20!white` est composée de 20% de vert et de 80% de blanc. Une telle expression de couleur est possible car PGF utilise l'extension `xcolor` de Uwe Kern. Voyez la documentation de cette extension pour des détails sur les expressions de couleur.

Que se serait-il passé si Karl n'avait pas « fermé » le chemin avec `--(0,0)` ? Dans ce cas le chemin aurait été fermé automatiquement et on aurait pu omettre le dernier point. En fait, il aurait mieux valu coder à la place quelque chose comme ce qui suit :

```
\fill[green!20!white] (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
```

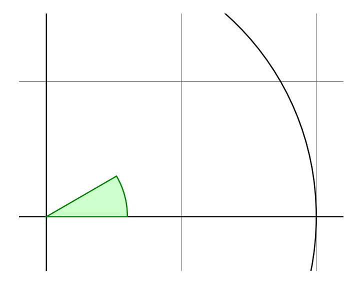
Le `--cycle` fait que le chemin courant est fermé (en fait la partie courante du chemin courant) en joignant le premier et le dernier point de façon plus lisse. Pour apprécier la différence regardez l'exemple suivant :

```
\begin{tikzpicture}[line width=5pt]
  \draw (0,0) -- (1,0) -- (1,1) -- (0,0);
  \draw (2,0) -- (3,0) -- (3,1) -- cycle;
  \useasboundingbox (0,1.5);
                % agrandit la boîte-cadre
\end{tikzpicture}
```

On peut tracer et remplir un chemin en même temps avec la commande `\filldraw`. Cela tracera d'abord le chemin puis le remplira. Cela peut ne pas sembler très utile mais on peut définir des couleurs différentes pour tracer et remplir. On spécifie cela avec des arguments optionnels comme ici :

```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
  \filldraw[fill=green!20!white, draw=green!50!black]
    (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\end{tikzpicture}
```



1.14. Estomper

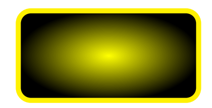
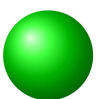
Karl considère brièvement la possibilité de rendre l'angle « plus fantaisiste » en l'*estompant*[2]. Au lieu de le remplir avec une couleur uniforme, on utilise une transition douce entre différentes couleurs. Pour cela on peut utiliser \shade ou \shadedraw, pour tracer et estomper en même temps :

```
\tikz \shade (0,0)
        rectangle (2,1)  (3,0.5)
        circle (.5cm);
```



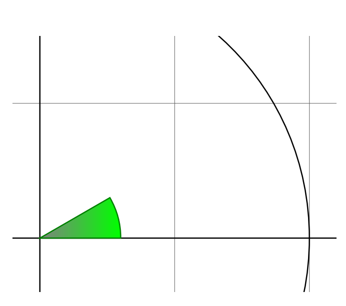
L'ombrage par défaut est une transition douce entre le gris et le blanc. Pour spécifier d'autres couleurs, on peut utiliser les options :

```
\begin{tikzpicture}[rounded corners,ultra thick]
 \shade[top color=yellow,bottom color=black]
                       (0,0) rectangle +(2,1);
 \shade[left color=yellow,right color=black]
                       (3,0) rectangle +(2,1);
 \shadedraw[inner color=yellow,outer color=black,draw=yellow]
                       (6,0) rectangle +(2,1);
 \shade[ball color=green] (9,.5) circle (.5cm);
\end{tikzpicture}
```

Pour Karl, ceci pourrait convenir :

```
\begin{tikzpicture}[scale=3]
 \clip (-0.1,-0.2) rectangle (1.1,0.75);
 \draw[step=.5cm,gray,very thin]
                  (-1.4,-1.4) grid (1.4,1.4);
 \draw (-1.5,0) -- (1.5,0);
 \draw (0,-1.5) -- (0,1.5);
 \draw (0,0) circle (1cm);
 \shadedraw[left color=gray,right color=green,
           draw=green!50!black]
   (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\end{tikzpicture}
```



2. Il faudrait, en toute rigueur, utiliser soit le verbe « dégrader » qui, d'après le Littré, s'applique à une couleur mais dont le sens courant est un peu trop prégnant et négatif pour qu'il ne soit pas source de confusion, soit la locution « appliquer un dégradé ». Je vais, une fois de plus, au plus court, revendiquant pour moi ce qu'on trouve bon pour la nature : le principe de moindre action. [N.D.T.]

Toutefois, il décide avec sagesse que l'estompage ne fait d'habitude que perturber le lecteur sans ajouter quoi que ce soit à la figure.

1.15. Définir des coordonnées

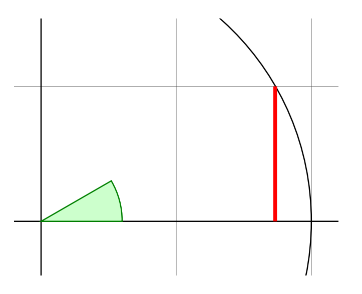
Karl veut maintenant ajouter les courbes du sinus et du cosinus. Il sait déjà qu'il peut définir la couleur de ces courbes avec l'option `color=`. Mais quelle est la meilleure manière de définir les coordonnées ?

Il y a deux manières de définir des coordonnées. La plus simple est d'écrire quelque chose comme `(10pt,2cm)`. Cela signifie 10pt suivant l'axe des x et 2 cm suivant l'axe des y. Autrement on peut aussi omettre les unités comme dans `(1,2)` qui signifie « une fois le vecteur courant suivant x plus deux fois le vecteur courant suivant y » [3]. Ces vecteurs ont pour longueur par défaut 1 cm dans chaque direction.

Afin de définir des points en coordonnées polaires on utilise la notation `(30:1cm)`, qui signifie 1 cm dans la direction 30 degrés. C'est évidemment bien utile pour « obtenir le point $(\cos 30°, \sin 30°)$ sur le cercle ».

On peut ajouter, devant les coordonnées, un signe `+` simple ou un double comme dans `+(1cm,0cm)` ou `++(0cm,2cm)`. De telles coordonnées sont interprétées différemment : la première signifie « 1 cm vers le haut depuis la position définie précédemment » et la deuxième « 2 cm vers la droite de la position précédemment définie en en faisant la nouvelle position définie ». Par exemple, on peut tracer la courbe du sinus comme suit :

```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin]
              (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
  \filldraw[fill=green!20,draw=green!50!black]
    (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
  \draw[red,very thick] (30:1cm) -- +(0,-0.5);
\end{tikzpicture}
```



3. pour un français frotté de maths du secondaire on dira « vecteur courant colinéaire au vecteur directeur de l'axe des abscisses » etc. [N.D.T.]

Karl utilise le fait que $\sin 30° = 1/2$. Toutefois, il doute très fortement que ses étudiants le sachent, aussi il pense que ce serait bien s'il y avait un moyen de déterminer un point comme « le point, juste sous le point `(30:1cm)`, placé sur l'axe des x ». C'est en fait possible avec une syntaxe spéciale : Karl peut écrire `(30:1cm|-0,0)`. De manière générale, le sens de `(`p `|-` q`)` est « le point d'intersection de la verticale passant par p et de l'horizontale passant par q ».

Ensuite, traçons la courbe du cosinus. Une façon serait d'écrire `(30:1cm|-0,0) -- (0,0)`. Une autre façon est la suivante : nous continuons depuis le point où s'arrête le sinus :

```
\begin{tikzpicture}[scale=3]
 \clip (-0.1,-0.2) rectangle (1.1,0.75);
 \draw[step=.5cm,gray,very thin]
                (-1.4,-1.4) grid (1.4,1.4);
 \draw (-1.5,0) -- (1.5,0);
 \draw (0,-1.5) -- (0,1.5);
 \draw (0,0) circle (1cm);
 \filldraw[fill=green!20,draw=green!50!black]
    (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
 \draw[red,very thick]  (30:1cm) -- +(0,-0.5);
 \draw[blue,very thick] (30:1cm)
                        ++(0,-0.5) -- (0,0);
\end{tikzpicture}
```

Notez qu'il n'y a pas de `--` entre `(30:1cm)` et `+(0,-0.5)`. En détail, le chemin est interprété comme suit : « d'abord, le `(30:1cm)` me dit de déplacer mon crayon jusque $(\cos 30°, 1/2)$. Ensuite vient une autre définition de coordonnées, aussi je déplace mon crayon sans rien tracer. Ce nouveau point est une unité sous la dernière position c.-à-d. à $(\cos 30°, 0)$. Enfin, je déplace le crayon à l'origine mais cette fois je trace quelque chose (à cause du `--`). »

Pour voir la différence entre + et ++ regardez ce qui suit :

```
\begin{tikzpicture}
  \def\rectanglepath{-- ++(1cm,0cm)  -- ++(0cm,1cm)
                     -- ++(-1cm,0cm) -- cycle}
  \draw (0,0) \rectanglepath;
  \draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

Par comparaison, lorsque l'on utilise un simple +, les coordonnées sont différentes :

```
\begin{tikzpicture}
  \def\rectanglepath{-- +(1cm,0cm) -- +(1cm,1cm)
                    -- +(0cm,1cm) -- cycle}
  \draw (0,0) \rectanglepath;
  \draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

Naturellement, on aurait pu écrire tout cela plus clairement et à moindre frais comme ceci (avec un simple ou un double +) :

```
\tikz \draw (0,0) rectangle +(1,1)
            (1.5,0) rectangle +(1,1);
```

Il reste à Karl le segment pour $\tan\alpha$ qu'il semble difficile de définir en utilisant les transformations et les coordonnées polaires. Pour cela il a besoin d'une autre façon de définir des coordonnées : Karl peut définir des coordonnées à l'aide d'intersections de droites. Le segment pour $\tan\alpha$ commence à $(1,0)$ et monte verticalement jusqu'au point d'intersection d'une droite verticale et d'une droite passant par l'origine et `(30:1cm)`. La syntaxe pour obtenir ce point est la suivante :

```
\draw[very thick,orange] (1,0) -- (intersection of 1,0--1,1 and 0,0--30:1cm);
```

Dans ce qui suit, deux derniers exemples présentent la façon d'utiliser les positions relatives. Notez que les options de transformations qui sont expliquées plus loin sont souvent plus utiles pour déplacer des objets que des positions relatives.

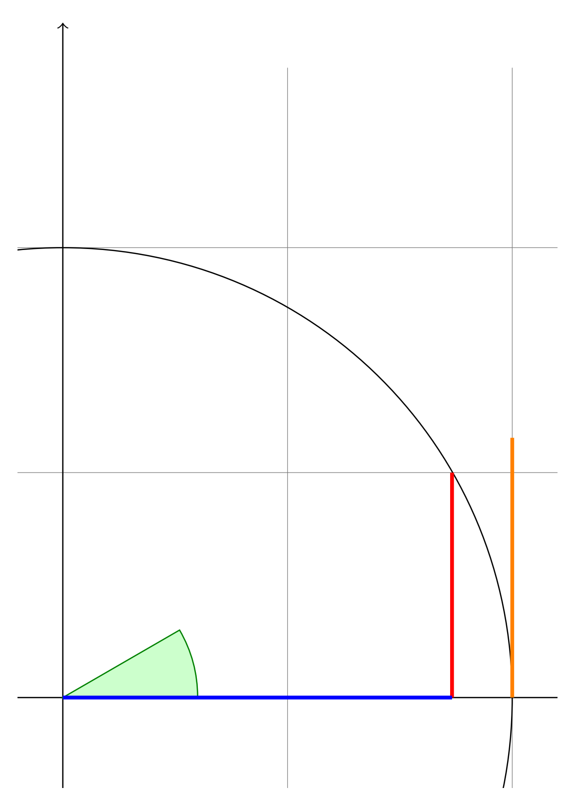
```
\begin{tikzpicture}[scale=0.5]
 \draw (0,0) -- (90:1cm) arc (90:360:1cm)
                         arc (0:30:1cm) -- cycle;
 \draw (60:5pt) -- +(30:1cm) arc (30:90:1cm) -- cycle;
 \draw (3,0)  +(0:1cm) -- +(72:1cm)
   -- +(144:1cm) -- +(216:1cm) -- +(288:1cm) -- cycle;
\end{tikzpicture}
```

1.16. Ajouter des pointes de flèches

Karl veut maintenant ajouter les petites pointes de flèche au bout des axes. Il a remarqué que dans de nombreuses figures, même dans des journaux scientifiques, ces pointes semblent manquer, peut-être parce que les programmes qui ont créé les graphiques ne peuvent pas les produire. Karl pense qu'il doit y avoir des pointes au bout des axes. Son fils est d'accord. Ses étudiants ne s'occupent pas des flèches.

Il se trouve qu'ajouter des flèches est plutôt simple : Karl ajoute l'option -> aux commandes de tracé des axes :

```
\begin{tikzpicture}[scale=5]
 \clip (-0.1,-0.2) rectangle (1.1,1.51);
 \draw[step=.5cm,gray,very thin]
                            (-1.4,-1.4) grid (1.4,1.4);
 \draw[->] (-1.5,0) -- (1.5,0);
 \draw[->] (0,-1.5) -- (0,1.5);
 \draw (0,0) circle (1cm);
 \filldraw[fill=green!20,draw=green!50!black]
            (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
 \draw[red,very thick]           (30:1cm) -- +(0,-0.5);
 \draw[blue,very thick]   (30:1cm) ++(0,-0.5) -- (0,0);
 \draw[orange,very thick]
   (1,0) -- (intersection of 1,0--1,1 and 0,0--30:1cm);
\end{tikzpicture}
```



Si Karl avait utilisé l'option <- à la place de -> les pointes auraient été placées au début du chemin. L'option <-> place des flèches à chaque extrémité du chemin.

Il y a certaines restrictions au type de chemins auxquels on peut ajouter des flèches. En gros, on ne peut ajouter des flèches qu'aux « lignes » simples ouvertes. Par exemple, on ne devrait pas essayer d'ajouter des flèches à, disons, un rectangle ou un cercle. (On peut essayer mais il n'y a aucune garantie sur ce qui arrivera ni aujourd'hui ni dans les prochaines versions.) Toutefois, on peut ajouter des flèches à des chemins courbes et à des chemins composés de plusieurs segments, comme dans les exemples qui suivent :

```
\begin{tikzpicture}
 \draw [<->] (0,0) arc (180:30:10pt);
 \draw [<->]
    (1,0) -- (1.5cm,10pt) -- (2cm,0pt) -- (2.5cm,10pt);
\end{tikzpicture}
```

Karl regarde plus en détail la flèche que Ti*k*Z place à la fin. Elle apparaît comme ceci quand il l'agrandit : $\rightarrow$. La forme semble vaguement familière et, de fait, c'est exactement celle du bout de la flèche normal que TEX utilise dans quelque chose comme $f\colon A \to B$.

Karl aime cette flèche d'autant plus qu'elle n'est pas aussi épaisse que celles offertes par tant d'extensions. Toutefois, il s'attend à ce que, parfois, il ait besoin d'utiliser d'autres genre de flèches. Pour cela, Karl peut écrire, par exemple `>=`*right arrow tip kind* où *right arrow tip kind* est une spécification spéciale de pointe de flèche. Par exemple, si Karl écrit `>=stealth` alors il dirait à Ti*k*Z qu'il voudrait une pointe de flèche « à la combattant furtif » :

```
\begin{tikzpicture}[>=stealth]
 \draw [->] (0,0) arc (180:30:10pt);
 \draw [<<-,very thick]
   (1,0) -- (1.5cm,10pt) -- (2cm,0pt) -- (2.5cm,10pt);
\end{tikzpicture}
```

Karl se demande si un nom aussi guerrier [4] pour un type de flèche est vraiment nécessaire. Il n'est pas vraiment radouci lorsque son fils lui dit que « PowerPoint » de Microsoft utilise le même nom. Il décide d'en faire discuter ses étudiants un jour.

En plus de `stealth` il y a de nombreuses autres pointes prédéfinies parmi lesquelles Karl peut choisir. De plus il peut définir des types de flèche lui-même s'il en a besoin.

4. L'adjectif « stealth » qui signifie furtif est employé pour un certain type d'avion militaire. [N.D.T.]

1.17. Portée

Karl a déjà vu que de nombreuses options de graphique peuvent influer sur le rendu des chemins. Souvent, il voudrait appliquer certaines options à tout un ensemble de commandes graphiques. Par exemple, Karl pourrait désirer tracer trois chemins en utilisant le crayon `thick` (*épais*) mais vouloir que le reste soit tracé « normalement ».

Si Karl souhaite qu'un certain ensemble d'options de graphique s'appliquent à toute la figure, il peut simplement passer ces options à la commande `\tikz` ou à l'environnement `{tikzpicture}` (Gerda passerait ces options à `\tikzpicture`). Toutefois, si Karl veut les appliquer à un groupe local, il place ces commandes dans un environnement `{scope}` (Gerda utilise `\scope` et `\endscope` et Hans utilise `\startscope` et `\stopscope`). Cet environnement prend des options de graphique comme argument optionnel et ces options s'appliquent à tout ce que contient l'environnement mais à rien d'extérieur.

Voici un exemple :

```
\begin{tikzpicture}[ultra thick]
  \draw (0,0) -- (0,1);
  \begin{scope}[thin]
    \draw (1,0) -- (1,1);
    \draw (2,0) -- (2,1);
  \end{scope}
  \draw (3,0) -- (3,1);
\end{tikzpicture}
```

Délimiter la portée à l'aide de `{scope}` a un autre effet intéressant : tout changement affectant la surface découpée est local à la portée. Ainsi si on écrit `\clip` quelque part dans la portée, l'effet de cette commande `\clip` s'achève à la fin de la portée. C'est utile car il n'y a pas d'autre moyen d'agrandir la partie découpée.

Karl a déjà vu également que les options données à une commande ne s'appliquent qu'à cette commande. Il se trouve que la situation est un peu plus compliquée. D'abord, les options passées à une commande comme `\draw` ne sont pas vraiment des options de la commande mais des « options de chemin » et peuvent être données n'importe où dans le chemin. Ainsi, au lieu de `\draw[thin] (0,0) -- (1,0);` on peut écrire aussi `\draw (0,0) [thin] -- (1,0);` ou `\draw (0,0) -- (1,0) [thin];`. Tous ces codes auront le même effet. Cela peut paraître étrange puisque dans le dernier cas il pourrait sembler que le `thin` ne prend effet qu'après que la droite de $(0,0)$ à $(1,0)$ a été tracée. Toutefois la plupart

des options de graphique ne s'applique qu'à un chemin complet. En fait si l'on écrit et `thin` et `thick` dans un même chemin, c'est la dernière des options données qui « gagne ».

En lisant ce qui précède, Karl remarque que seulement « la plupart » des options de graphique s'applique au chemin complet. En fait, toutes les options de transformations *ne* s'appliquent *pas* au chemin entier mais seulement à « tout ce qui les suit sur le chemin ». Nous regarderons cela plus en détail d'ici peu. Cependant toutes les options données pendant la construction d'un chemin ne s'appliquent qu'à ce chemin.

1.18. Transformations

Quand on définit des coordonnées comme `(1cm,1cm)`, où le point sera-t-il placé sur la page ? Pour déterminer cette position, Ti*k*Z, TeX et PDF ou PostScript appliquent tous certaines transformations aux coordonnées données pour déterminer la position finale sur la page.

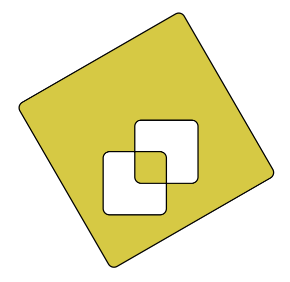
Ti*k*Z fournit de nombreuses options qui permettent de transformer les coordonnées dans le système privé de coordonnées de PGF. Par exemple, l'option `xshift` permet de pousser tous les points suivants d'une certaine quantité :

```
\tikz \draw (0,0) -- (0,0.5) [xshift=2pt] (0,0) -- (0,0.5);
```

Il est important de noter que l'on peut changer de transformation « au milieu du chemin », une caractéristique que n'a ni PDF ni PostScript. La raison en est que PGF garde la trace de sa propre matrice de transformation.

Voici un exemple plus complexe :

```
\begin{tikzpicture}[even odd rule,rounded
                              corners=2pt,x=10pt,y=10pt,scale=2]
 \filldraw[fill=examplefill] (0,0)   rectangle (1,1)
   [xshift=5pt,yshift=5pt]   (0,0)   rectangle (1,1)
               [rotate=30]   (-1,-1) rectangle (2,2);
\end{tikzpicture}
```



Les transformations les plus utiles sont `xshift` et `yshift` pour déplacer, `shift` pour déplacer jusqu'à un point donné comme dans `shift={(1,0)}` ou `shift={+(0,0)}` (les accolades sont nécessaires pour que TEX ne prenne pas les virgules comme séparateurs d'options), `rotate` pour faire tourner d'un certain angle (il y a également un `rotate around` pour faire tourner autour d'un point donné), `scale` pour agrandir ou rétrécir d'un certain facteur, `xscale` et `yscale` pour agrandir uniquement parallèlement à l'axe des x ou l'axe des y, (`xsacle=-1` est une symétrie axiale) et `xslant` et `yslant` pour incliner. Si ces transformations et celles que je n'ai pas mentionnées ne suffisent pas, l'option `cm` permet d'appliquer une matrice de transformation arbitraire. Les étudiants de Karl, au passage, ne savent pas ce qu'est une matrice de transformation.

1.19. Répéter : boucles pour

Le but suivant de Karl est d'ajouter des graduations sur les axes aux emplacements -1, $-1/2$, $1/2$ et 1. Pour ce faire il lui serait agréable de pouvoir utiliser une sorte de « boucle » d'autant plus qu'il souhaite faire la même chose à chacune de ces positions. Il existe plusieurs extensions pour faire ça. LATEX a ses propres commandes internes, `pstricks` est livré avec la puissante commande `\multido`. Toutes peuvent être utilisées avec PGF et Ti*k*Z, aussi si vous en êtes familier, n'hésitez pas à les utiliser. PGF apporte encore une autre commande, nommée `\foreach` (*pour chaque*), que j'ai ajoutée parce que je n'arrivais jamais à me souvenir de la syntaxe des autres extensions. `\foreach` est définie dans l'extension `pgffor` et peut être utilisée indépendamment de PGF. Ti*k*Z la charge automatiquement.

Dans sa forme de base, la commande `\foreach` est d'une utilisation aisée :

```
\foreach \x in {1,2,3} {$x =\x$, } $x=4$.
```

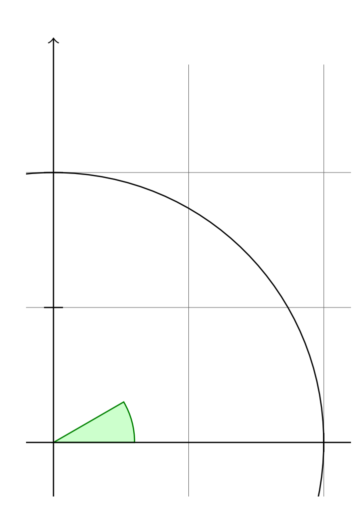
$$x = 1, x = 2, x = 3, \; x = 4.$$

La syntaxe générale de cette macro est `\foreach` *variable* `in {`*liste de valeurs*`}` *commandes*. À l'intérieur de *commandes* la *variable* prendra les diverses valeurs. Si *commandes* ne commence pas par une accolade, tout ce qui précède le prochain point-virgule est utilisé comme *commandes*.

Pour Karl et les graduations sur les axes, il pourrait utiliser le code suivant :

```
\begin{tikzpicture}[scale=3]
 \clip (-0.1,-0.2) rectangle (1.1,1.51);
 \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
 \filldraw[fill=green!20,draw=green!50!black]
                (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
 \draw[->] (-1.5,0) -- (1.5,0);
 \draw[->] (0,-1.5) -- (0,1.5);
 \draw (0,0) circle (1cm);

 \foreach \x in {-1cm,-0.5cm,1cm}
      \draw (\x,-1pt) -- (\x,1pt);
 \foreach \y in {-1cm,-0.5cm,0.5cm,1cm}
      \draw (-1pt,\y) -- (1pt,\y);
\end{tikzpicture}
```



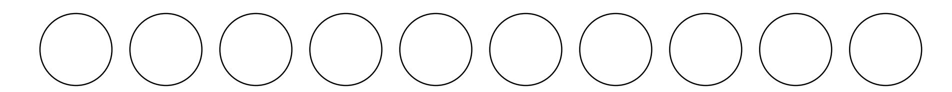
En fait, il y a plusieurs façon de créer les graduations. Par exemple, Karl pourrait avoir placé le `\draw ...;` entre accolades. Il pourrait aussi avoir utilisé, par exemple,

```
\foreach \x in {-1,-0.5,1}
  \draw[xshift=\x cm] (0pt,-1pt) -- (0pt,1pt);
```

Karl est curieux de savoir ce qui se passerait dans une situation plus complexe lorsqu'il y a, disons, 20 graduations. Cela semble ennuyeux d'avoir à mentionner explicitement tous les nombres dans l'ensemble de `\foreach`. En effet, on peut utiliser `...` dans l'expression du `\foreach`

pour itérer sur un grand nombre de valeurs (qui doivent être cependant des nombres réels sans dimension) comme dans l'exemple suivant :

```
\tikz \foreach \x in {1,...,10}
        \draw (\x,0) circle (0.4cm);
```



Si on fournit *deux* nombres avant le `...`, l'expression du `\foreach` utilisera leur différence comme pas :

```
\tikz \foreach \x in {-1,-0.5,...,1}
       \draw (\x cm,-1pt) -- (\x cm,1pt);
```

On peut aussi emboîter les boucles pour créer des effets intéressants :

```
\begin{tikzpicture}[scale=.8]
  \foreach \x in {1,2,...,5,7,8,...,12}
    \foreach \y in {1,...,5}
    {
      \draw (\x,\y) +(-.5,-.5) rectangle ++(.5,.5);
      \draw (\x,\y) node{\x,\y};
    }
\end{tikzpicture}
```

1,5	2,5	3,5	4,5	5,5
1,4	2,4	3,4	4,4	5,4
1,3	2,3	3,3	4,3	5,3
1,2	2,2	3,2	4,2	5,2
1,1	2,1	3,1	4,1	5,1

7,5	8,5	9,5	10,5	11,5	12,5
7,4	8,4	9,4	10,4	11,4	12,4
7,3	8,3	9,3	10,3	11,3	12,3
7,2	8,2	9,2	10,2	11,2	12,2
7,1	8,1	9,1	10,1	11,1	12,1

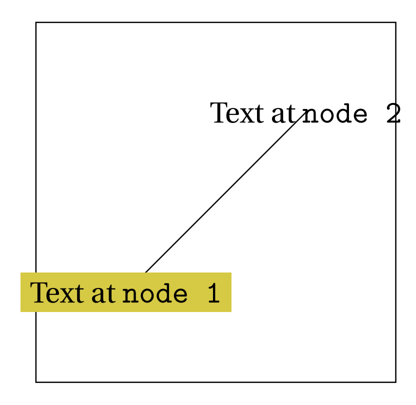
L'expression `\foreach` peut faire des choses encore plus sournoises mais ce qui précède donne l'idée principale.

1.20. Ajouter du texte

Karl est, maintenant, assez satisfait de la figure. Toutefois, la partie la plus importante, à savoir les étiquettes, manque encore !

Ti*k*Z offre un système puissante et facile à utiliser pour ajouter à une figure du texte et, plus généralement, des formes complexes à une position donnée. L'idée de base est la suivante : lorsque Ti*k*Z construit un chemin et rencontre le mot-clef `node` (*nœud*) au milieu du chemin, il lit la *spécification de nœud*. Le mot-clef `node` est suivi de quelques options et de texte placé entre accolades. Ce texte est placé dans une boite normale de TEX (si la spécification de nœud suit directement des coordonnées, ce qui est le cas généralement, Ti*k*Z est capable de faire un peu de magie afin qu'il soit même possible d'utiliser du texte *verbatim* dans les boites) et puis la place à la position courante, c.-à-d. à la dernière position déterminée (éventuellement déplacée un peu, suivant les options données). Toutefois, tous les nœuds ne sont tracés seulement qu'après que le chemin a été complètement tracé/rempli/ombré/découpé/etc.

```
\begin{tikzpicture}[scale=2]
  \draw (0,0) rectangle (2,2);
  \draw (0.5,0.5) node [fill=examplefill]
                       {Text at \verb!node 1!}
     -- (1.5,1.5) node {Text at \verb!node 2!};
\end{tikzpicture}
```



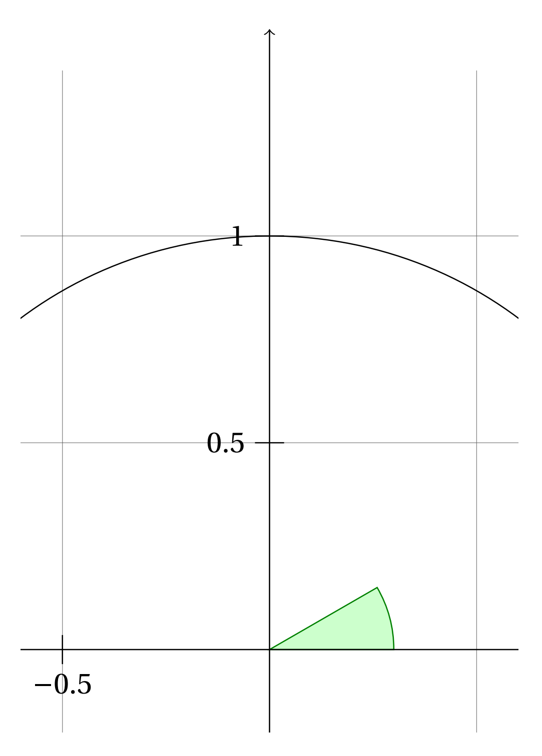
Visiblement, Karl voudrait placer des nœuds non seulement à la dernière position définie mais aussi à gauche ou à droite de cette position. Pour cela, chaque nœud qu'on place dans une figure est équipé de plusieurs *ancres* (*anchor*). Par exemple, l'ancre `north` (*nord*) est au milieu à la partie supérieure de la forme, l'ancre `south` (*sud*) est au bas et

l'ancre `north east` (*nord-est*) est dans le coin supérieur droit. Lorsque l'on passe l'option `anchor=north`, le texte est placé de telle sorte que cette ancre nordique est située à la position courante et que le texte est, de ce fait, sous la position courante. Karl utilise cela pour tracer les graduations comme suit :

```
\begin{tikzpicture}[scale=4.6]
  \clip (-0.6,-0.2) rectangle (0.6,1.51);
  \draw[step=.5cm,style=help lines] (-1.4,-1.4) grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black]
    (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
  \draw[->] (-1.5,0) -- (1.5,0);   \draw[->] (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);

  \foreach \x in {-1,-0.5,1}
    \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {$\x$};

  \foreach \y in {-1,-0.5,0.5,1}
    \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east] {$\y$};
\end{tikzpicture}
```



C'est déjà pas mal. En utilisant ces ancres, Karl peut maintenant ajouter la plupart des autres éléments textuels. Toutefois Karl pense

que, bien que « correcte », c'est plutôt contre-intuitif que, pour placer quelque chose *sous* un point donné, il faille utiliser l'ancre *north.* Pour cette raison, il y a une option `below` (*dessous*) qui produit la même chose que `anchor=north`. De même, `above right` (*dessus à droite*) fait comme `anchor=south east`. De plus `below` prend un argument optionnel de dimension. Si on le donne, la forme sera, en plus, poussée vers le bas de la valeur donnée. Ainsi, on peut utiliser `below=1pt` pour placer une étiquette textuelle sous un point et, en plus, la déplacer de 1pt vers le bas.

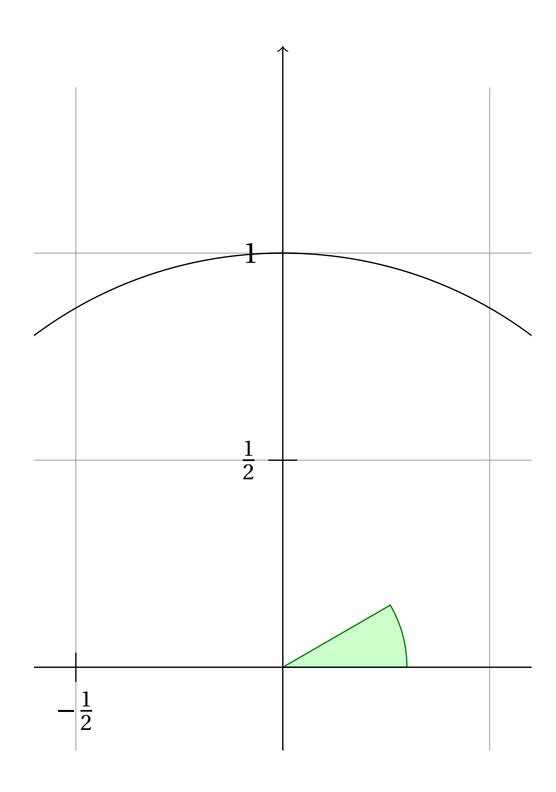
Karl n'est pas tout à fait satisfait des graduations. Il voudrait avoir 1/2 ou $\frac{1}{2}$ plutôt que 0.5, en partie pour faire montre des exquises capacités de TEX et de Ti*k*Z, en partie parce que pour des positions comme 1/3 ou π il est de beaucoup préférable d'avoir la graduation « mathématique » plutôt que juste « numérique ». Ces étudiants, par ailleurs, préfèrent 0.5 à 1/2 puisqu'ils ne sont pas de manière générale très friands de fractions.

Karl maintenant fait face à un problème : dans l'expression `\foreach` la position `\x` devrait toujours être données comme `0.5` puisque Ti*k*Z ne comprendrait pas ce que `\frac{1}{2}` est sensé être. Par ailleurs, le texte écrit devrait vraiment être `\frac{1}{2}`. Pour résoudre ce problème, `\foreach` offre une syntaxe spéciale : au lieu d'une seule variable `\x`, Karl peut en définir deux (ou même plus) séparées par une barre oblique (*slash*) comme dans `\x / \xtext`. Alors, les éléments de l'ensemble sur lequel `\foreach` itère doivent être aussi de la forme *premier/second.* À chaque itération, `\x` prendra la valeur de *premier* et `\xtext` celle de *second.* Si on ne donne pas de *second*, le *premier* est utilisé à nouveau. Aussi, voici le nouveau code pour les graduations :

```
\begin{tikzpicture}[scale=4.6]
 \clip (-0.6,-0.2) rectangle (0.6,1.51);
 \draw[step=.5cm,style=help lines] (-1.4,-1.4) grid (1.4,1.4);
 \filldraw[fill=green!20,draw=green!50!black]
   (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
 \draw[->] (-1.5,0) -- (1.5,0); \draw[->] (0,-1.5) -- (0,1.5);
 \draw (0,0) circle (1cm);

 \foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
   \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {$\xtext$};

 \foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
    \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east] {$\ytext$};
\end{tikzpicture}
```



Le résultat fait plaisir à Karl mais son fils montre que ce n'est pas parfaitement satisfaisant : le quadrillage et le cercle interfèrent avec les nombres et amoindrissent leur lisibilité. Karl ne se sent pas très concerné par ça (ses étudiants ne s'en rendent même pas compte) mais son fils insiste sur le fait qu'il y a une solution facile : Karl peut ajouter l'option `[fill=white]` pour remplir l'arrière plan de la forme de texte de blanc.

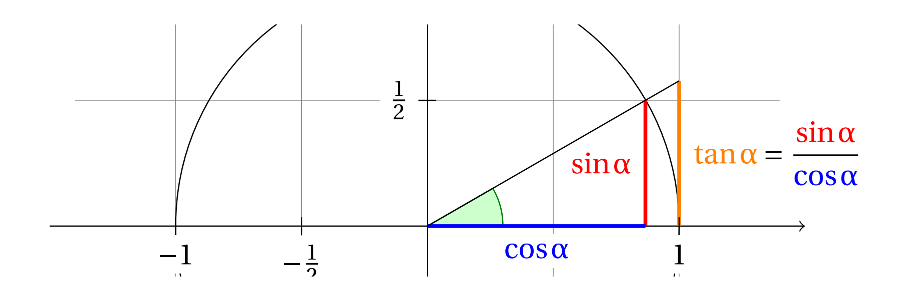
Karl veut ensuite ajouter des étiquettes comme $\sin\alpha$. Celles-là il voudrait les placer « au milieu de la courbe ». Pour ce faire, au lieu de définir l'étiquette `node {$\sin\alpha$}` directement après une des extrémités de la courbe (ce qui placerait l'étiquette à cette extrémité), Karl peut placer cette étiquette directement après `--` mais avant les coordonnées. Par défaut, cela place l'étiquette au milieu de la courbe mais on peut utiliser les options `pos=` pour modifier ce comportement. On peut également utiliser les options comme `near start` et `near end` pour changer la position.

```
\begin{tikzpicture}[scale=2.8]
 \clip (-2,-0.2) rectangle (2,0.8);
 \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
 \filldraw[fill=green!20,draw=green!50!black]
        (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
 \draw[->] (-1.5,0) -- (1.5,0) coordinate (x axis);
 \draw[->] (0,-1.5) -- (0,1.5) coordinate (y axis);
 \draw (0,0) circle (1cm);

 \draw[very thick,red]
   (30:1cm) -- node[left=1pt,fill=white]
                {$\sin \alpha$} (30:1cm |- x axis);
 \draw[very thick,blue]
   (30:1cm |- x axis) -- node[below=2pt,fill=white]
                             {$\cos \alpha$} (0,0);
 \draw[very thick,orange]
   (1,0) -- node [right=1pt,fill=white]
   {$\displaystyle \tan \alpha \color{black}=
     \frac{{\color{red}\sin \alpha}}{\color{blue}\cos \alpha}$}
   (intersection of 0,0--30:1cm and 1,0--1,1) coordinate (t);

 \draw (0,0) -- (t);

 \foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
   \draw (\x cm,1pt) -- (\x cm,-1pt)
                    node[anchor=north,fill=white] {$\xtext$};
 \foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
   \draw (1pt,\y cm) -- (-1pt,\y cm)
                     node[anchor=east,fill=white] {$\ytext$};
\end{tikzpicture}
```



On peut placer les étiquettes sur les courbes et, en ajoutant l'option `sloped` (*déclive, en pente*), faire qu'elles soient tournées de telle sorte

qu'elles suivent la pente de la courbe. Voici un exemple :

```
\begin{tikzpicture}
  \draw (0,0) .. controls (5,1) and (7.5,1) ..
    node[near start,sloped,above] {near start}
    node {midway}
    node[very near end,sloped,below] {very near end} (10,0);
\end{tikzpicture}
```

near start midway very near end

Il reste à écrire le texte explicatif sur la droite de la figure. La difficulté principale ici est de limiter la largeur de « l'étiquette textuelle » qui est assez longue ce qui pousse à utiliser la coupure de ligne. Heureusement, Karl peut utiliser l'option `text width=6cm` pour obtenir l'effet voulu. Ainsi, voici le code complet :

```
\begin{tikzpicture}[scale=3,cap=round]
 % Local definitions
 \def\costhirty{0.8660256}

 % Colors
 \colorlet{anglecolor}{green!50!black}
 \colorlet{sincolor}{red}
 \colorlet{tancolor}{orange!80!black}
 \colorlet{coscolor}{blue}

 % Styles
 \tikzstyle{axes}=[]
 \tikzstyle{important line}=[very thick]
 \tikzstyle{information text}=
   [rounded corners,fill=red!10,inner sep=1ex]

 % The graphic
 \draw[style=help lines,step=0.5cm] (-1.4,-1.4) grid (1.4,1.4);

 \draw (0,0) circle (1cm);

 \begin{scope}[style=axes]
  \draw[->] (-1.5,0) -- (1.5,0)
    node[right] {$x$} coordinate(x axis);
  \draw[->] (0,-1.5) -- (0,1.5)
    node[above] {$y$} coordinate(y axis);

  \foreach \x/\xtext in {-1, -.5/-\frac{1}{2}, 1}
   \draw[xshift=\x cm] (0pt,1pt) -- (0pt,-1pt)
```

```
    node[below,fill=white] {$\xtext$};

  \foreach \y/\ytext in {-1, -.5/-\frac{1}{2}, .5/\frac{1}{2}, 1}
   \draw[yshift=\y cm] (1pt,0pt) -- (-1pt,0pt)
     node[left,fill=white] {$\ytext$};
 \end{scope}

 \filldraw[fill=green!20,draw=anglecolor]
   (0,0) -- (3mm,0pt) arc(0:30:3mm);
 \draw (15:2mm) node[anglecolor] {$\alpha$};

 \draw[style=important line,sincolor]
  (30:1cm) -- node[left=1pt,fill=white]
  {$\sin \alpha$} (30:1cm |- x axis);

 \draw[style=important line,coscolor]
  (30:1cm |- x axis) -- node[below=2pt,fill=white]
  {$\cos \alpha$} (0,0);

 \draw[style=important line,tancolor] (1,0) --
 node[right=1pt,fill=white] {
  $\displaystyle \tan \alpha \color{black}=
  \frac{{\color{sincolor}\sin \alpha}}{\color{coscolor}\cos \alpha}$}
  (intersection of 0,0--30:1cm and 1,0--1,1) coordinate (t);

 \draw (0,0) -- (t);

 \draw[xshift=1.85cm]
  node[right,text width=6cm,style=information text]
  {
   L'{\color{anglecolor} angle $\alpha$} vaut 30\degre\
   dans l'exemple ($\pi/6$ en radians). Le
   {\color{sincolor}sinus de $\alpha$}, qui est la
   longueur du segment rouge est
   \[{\color{sincolor}\sin\alpha} = 1/2.\]
   Par le théorème de Pythagore, ...
  };
\end{tikzpicture}
```

2. Un réseau de Petri pour Hagen

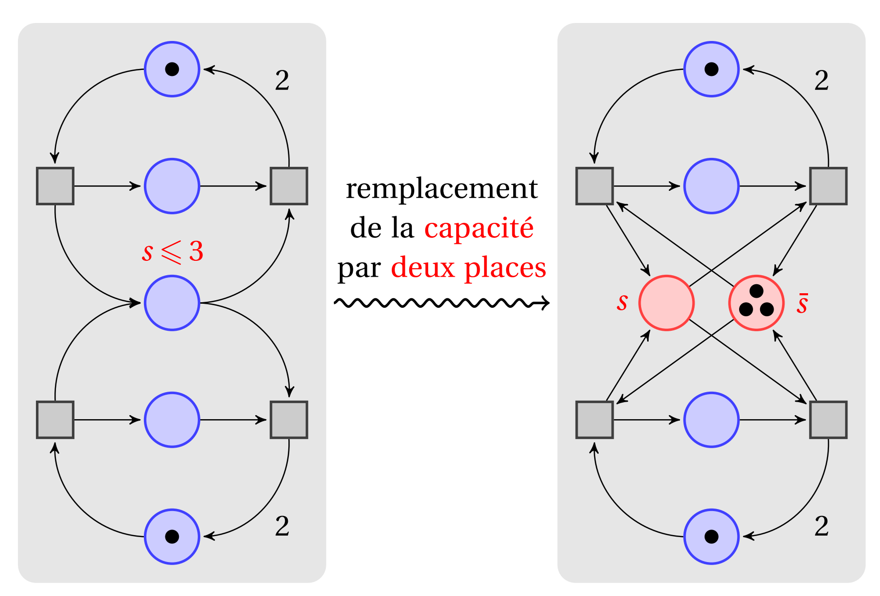
Dans cette seconde partie, nous explorerons les méchanismes des nœuds pour Ti*k*Z et PGF.

Hagen doit faire demain un exposé sur son formalisme favori pour les systèmes distribués : les réseaux de Petri ! Hagen avait l'habitude de faire des exposés en utilisant le tableau et tout le monde semblait parfaitement content de cette façon. Malheureusement, on a récemment gâté son public avec des présentations sur rétro-projecteurs et il semble

y avoir une certaine pression de ses pairs pour que ces réseaux de Petri soient, eux aussi, dessinés avec un logiciel. Un de ses professeurs à l'institut lui recommande Ti*k*Z et Hagen décide de tenter le coup.

2.1. Énoncé du problème

Hagen, dans son exposé, veut créer un graphique qui montrerait comment un réseau de Petri à capacité peut être émulé par un réseau de Petri ordinaire. Idéalement, le graphique devrait ressembler à ceci :



2.2. Mettre en place l'environnement

Pour dessiner, Hagen doit charger l'extension Ti*k*Z comme Karl l'a fait dans la première partie. Toutefois, Hagen aura besoin de charger aussi quelques *extension de bibliothèques* supplémentaires. Ces extensions contiennent des définitions additionnelles comme des pointes de flèches que l'on n'utilise pas en général dans un dessin et que l'on doit charger explicitement.

Hagen doit charger trois bibliothèques : la bibliothèque de pointes de flèches pour obtenir la pointe de flèche spéciale utilisée dans le graphique, la bibliothèque de serpent pour la « ligne serpentante » au milieu et la bibliothèque d'arrière-plans pour obtenir les deux surfaces rectangulaires placées derrière les deux parties principales de la figure.

2.2.1. *Mettre en place l'environnement dans LaTeX*

Quand on utilise LaTeX, on écrira

```
\documentclass{article} %par exemple
\usepackage{tikz}
\usetikzlibrary{arrows,snakes,backgrounds}
\begin{document}
  \begin{tikzpicture}
    \draw (0,0) -- (1,1);
  \end{tikzpicture}
\end{document}
```

2.2.2. *Mettre en place l'environnement dans Plain TeX*

Avec plain TeX, on utilisera

```
%% Plain TeX file
\input tikz.tex
\usetikzlibrary{arrows,snakes,backgrounds}
\baselineskip=12pt
\hsize=6.3truein
\vsize=8.7truein
\tikzpicture
  \draw (0,0) -- (1,1);
\endtikzpicture
\bye
```

2.2.3. *Mettre en place l'environnement dans ConTeXt*

Avec ConTeXt, on utilisera

```
%% ConTeXt file
\usemodule[tikz]
\usetikzlibrary[arrows,snakes,backgrounds]

\starttext
  \starttikzpicture
    \draw (0,0) -- (1,1);
  \stoptikzpicture
\startext
```

2.3. Introduction aux nœuds

En principe nous savons déjà créer la figure que désire Hagen (sauf peut-être la ligne serpentante, on va s'en occuper) : on commence avec un grand rectangle gris clair et puis on ajoute des tas de cercles et de petits rectangles, plus quelques flèches.

Toutefois cette approche a de nombreux défauts : premièrement, il sera difficile de changer quoi que ce soit à une étape ultérieure. Par exemple, si on décide d'ajouter plusieurs places au réseau de Petri (les

cercles sont appelés « places » dans la théorie des réseaux de Petri), toutes les coordonnées changent et on doit tout recalculer. Deuxièmement, il est difficile de lire le code du réseau de Petri puisque c'est une longue liste compliquée de coordonnées et de commandes de dessin ; la structure du réseau de Petri sous-jacent est perdue.

Heureusement, Ti*k*Z offre un mécanisme puissant pour éviter les difficultés énoncées ci-dessus : les nœuds. Nous avons déjà abordé les nœuds dans la partie précédente quand nous nous en sommes servi pour ajouter des étiquettes au graphique de Karl. Dans cette partie-ci, nous verrons que les nœuds sont bien plus puissants encore.

Un nœud est une petite partie d'une figure. Quand on crée un nœud, on fournit une position où le nœud devrait être dessiné et une *forme* (shape). Un nœud de forme `circle` (*cercle*) sera dessiné comme un cercle, un nœud de forme `rectangle` sera dessiné comme un rectangle, et ainsi de suite. Un nœud peut aussi contenir du texte, ce pourquoi Karl a utilisé des nœuds pour afficher du texte. Enfin, un nœud peut avoir un *nom* (name) pour référence ultérieure.

Dans la figure de Hage nous utiliserons des nœuds pour les places et les transitions du réseau de Petri (les places sont les cercles et les transitions les rectangles). Commençons par la moitié supérieure du réseau de Petri de gauche. Dans cette moitié supérieure nous avons trois places et deux transitions. Plutôt que de dessiner trois cercles et deux rectangles, nous utiliserons trois nœuds de forme `circle` et deux nœuds de forme `rectangle`.

```
\begin{tikzpicture}
  \path ( 0,2) node [shape=circle,draw] {}
        ( 0,1) node [shape=circle,draw] {}
        ( 0,0) node [shape=circle,draw] {}
        ( 1,1) node [shape=rectangle,draw] {}
        (-1,1) node [shape=rectangle,draw] {};
\end{tikzpicture}
```

Hagen note que cela ne ressemble pas vraiment à la figure finale mais que ça semble un bon premier pas.

Regardons le code plus attentivement. La figure complète consiste en un seul et unique chemin. En ignorant les opérations `node`, on ne trouve pas grand-chose dans ce chemin : il s'agit simplement d'une suite de coordonnées sans que rien « n'arrive » entre elles. En fait, même si quelque chose devait arriver, comme un « line-to » ou un « curve-to », la

commande \path ne « ferait » rien du chemin résultant. Aussi, toute la magie doit être dans les commandes node.

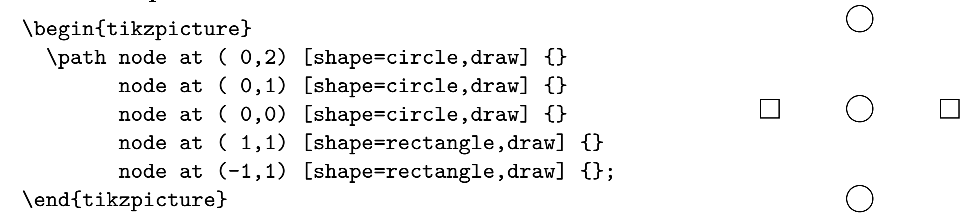
Dans la première partie, nous avons appris que chaque nœud ajoutera un morceau de texte à la dernière coordonnée. Ainsi, chacun des cinq nœuds est ajouté à une position différente. Dans le code précédent le texte est vide (à cause des {} vides). Alors pourquoi voyons-nous quelque chose ? La réponse est dans l'option draw de l'opération node : elle fait que « la forme autour du texte » est dessinée.

Donc, le code (0,2) node [shape=circle,draw] {} signifie : « dans le chemin principal, ajouter un move-to à la coordonnée (0,2). Puis suspendre temporairement la construction du chemin principal pendant que le nœud est construit. Ce nœud sera un circle (*cercle*) autour d'un texte vide. Ce cercle doit être dessiné (draw) mais ni rempli ni utilisé à autre chose. Lorsque le nœud tout entier est construit, le sauvegarder jusqu'à ce que le chemin principal soit terminé. Alors, le dessiner. » Le (0,1) node [shape=circle,draw] {} qui suit signifie : « Continuer le chemin principal par un move-to jusqu'à (0,1). Construire ensuite un nœud à cette position également. Ce nœud sera montré après que le chemin principal sera fini. » Et ainsi de suite.

2.4. Placer des nœuds grace à la syntax « at »

Hagen comprend maintenant que les opérations node ajoutent des nœuds au chemin mais il lui semble un rien stupide de créer, avec l'opération \path, un chemin, constitué d'opération move-to superflues, juste pour placer des nœuds. Il est heureux d'apprendre qu'il y a d'autres manières d'ajouter des nœuds d'une façon plus raisonnable.

D'abord, l'opération node nous autorise à écrire at (*coordinate*) afin de définir directement où le nœud doit être placé en rendant caduque la règle qui veut qu'il soit placé à la dernière coordonnée. Hagen peut donc écrire ce qui suit :



```
\begin{tikzpicture}
  \path node at ( 0,2) [shape=circle,draw] {}
        node at ( 0,1) [shape=circle,draw] {}
        node at ( 0,0) [shape=circle,draw] {}
        node at ( 1,1) [shape=rectangle,draw] {}
        node at (-1,1) [shape=rectangle,draw] {};
\end{tikzpicture}
```

Maintenant Hagen reste encore avec un simple chemin vide mais, au

moins, sans étrange `move-to`. Il se trouve que l'on peut encore apporter une amélioration ; la commande `\node` est une abréviation pour `\path node` ce qui permet à Hagen d'écrire :

```
\begin{tikzpicture}
  \node at ( 0,2) [circle,draw] {};
  \node at ( 0,1) [circle,draw] {};
  \node at ( 0,0) [circle,draw] {};
  \node at ( 1,1) [rectangle,draw] {};
  \node at (-1,1) [rectangle,draw] {};
\end{tikzpicture}
```

Hagen préfère cette syntaxe à la précédente. Notez que Hagen a également omis les `shape=` car, de même que pour les `color=`, Ti*k*Z permet d'omettre les `shape=` s'il n'y a pas de confusion possible.

2.5. Utiliser les styles

D'humeur aventureuse, Hagen essaie d'embellir les nœuds. Dans la figure finale, les cercles et rectangles devraient être remplis avec des couleurs différentes, cela conduit au code suivant :

```
\begin{tikzpicture}[thick]
  \node at ( 0,2) [circle,draw=blue!50,fill=blue!20] {};
  \node at ( 0,1) [circle,draw=blue!50,fill=blue!20] {};
  \node at ( 0,0) [circle,draw=blue!50,fill=blue!20] {};
  \node at ( 1,1) [rectangle,
                   draw=black!50,fill=black!20] {};
  \node at (-1,1) [rectangle,
                   draw=black!50,fill=black!20] {};
\end{tikzpicture}
```

Bien que la figure soit jolie, le code commence à devenir affreux. Idéalement, nous voudrions que notre code transporte le message « il y a trois places et deux transitions » et pas vraiment quelle couleur de fond doit être utilisée.

Pour résoudre ce problème, Hagen utilise des styles. Il définit un style pour les places et un autre pour les transitions :

```
\tikzstyle{place}=[circle,draw=blue!50,fill=blue!20,thick]
\tikzstyle{transition}=[rectangle,
                   draw=black!50,fill=black!20,thick]
\begin{tikzpicture}
  \node at ( 0,2) [place] {};
  \node at ( 0,1) [place] {};
  \node at ( 0,0) [place] {};
  \node at ( 1,1) [transition] {};
  \node at (-1,1) [transition] {};
\end{tikzpicture}
```

Avant que Hagen commence à nommer et connecter les nœuds, assurons-nous d'abord qu'ils aient leur apparence finale. Ils sont encore trop petits. En fait, Hagen se demande pourquoi ils ont une taille, après tout, le texte est vide. La raison en est que Ti*k*Z ajoute automatiquement de l'espace autour du texte. La quantité en est déterminée à l'aide de l'option `inner sep`. Aussi, pour accroître la taille des nœuds, Hagen pourrait écrire :

```
\tikzstyle{place}=[circle,
               draw=blue!50,fill=blue!20,thick]
\tikzstyle{transition}=[rectangle,
               draw=black!50,fill=black!20,thick]
\begin{tikzpicture}[inner sep=2mm]
  \node at ( 0,2) [place] {};
  \node at ( 0,1) [place] {};
  \node at ( 0,0) [place] {};
  \node at ( 1,1) [transition] {};
  \node at (-1,1) [transition] {};
\end{tikzpicture}
```

Toutefois, ce n'est pas vraiment la meilleure manière d'obtenir l'effet voulu. Il vaut bien mieux utiliser l'option `minimum size`. Cette option autorise Hagen à spécifier une taille minimale pour le nœud. Si les nœuds doivent en fait être plus gros du fait d'un texte plus long ils s'agrandiront mais si le texte est vide alors le nœud aura pour taille `minimum size` (*taille minimale*). Cette option est aussi utile pour s'assurer que plusieurs nœuds contenant différentes quantités de texte auront la même taille. Les options `minimum height` (*hauteur minimale*) et `minimum width` (*largeur minimale*) permettent de spécifier une hauteur et une largeur minimale de manière indépendante.

Aussi ce dont Hagen a besoin c'est de fournir une `minimum size` pour les nœuds. Par acquis de conscience, il définit également `inner sep=0pt`. Ainsi il s'assure que les nœuds auront vraiment la taille `minimum size` et non, pour des tailles minimales très petites, la taille minimale nécessaire pour contenir l'espace ajouté automatiquement.

```
\tikzstyle{place}=[circle,
                    draw=blue!50,fill=blue!20,thick,
                    inner sep=0pt,minimum size=6mm]
\tikzstyle{transition}=[rectangle,
                   draw=black!50,fill=black!20,thick,
                   inner sep=0pt,minimum size=4mm]
\begin{tikzpicture}
  \node at ( 0,2) [place] {};
  \node at ( 0,1) [place] {};
  \node at ( 0,0) [place] {};
  \node at ( 1,1) [transition] {};
  \node at (-1,1) [transition] {};
\end{tikzpicture}
```

2.7. Nommer les nœuds

Le prochain but de Hagen est de joindre les nœuds à l'aide de flèches. Ça semble être un problème embêtant puisque les flèches ne doivent pas commencer au milieu des nœuds mais quelque part sur la frontière et Hagen voudrait bien ne pas avoir à calculer ces positions à la main.

Heureusement, PGF fera les calculs nécessaires pour lui. Toutefois, il devra d'abord donner des noms aux nœuds afin de pouvoir y référence plus tard.

Il y a deux façons de nommer un nœud. La première utilise l'option `name=`. La seconde consiste à écrire le nom voulu entre parenthèse après l'opération `node`. Hagen pense que cette seconde méthode semble étrange mais il changera bientôt d'opinion.

```
% ... définition des styles
\begin{tikzpicture}
  \node (waiting 1)  at ( 0,2)     [place] {};
  \node (critical 1) at ( 0,1)     [place] {};
  \node (semaphore)  at ( 0,0)     [place] {};
  \node (leave critical) at ( 1,1) [transition] {};
  \node (enter critical) at (-1,1) [transition] {};
\end{tikzpicture}
```

Hagen est content de noter que les noms facilitent la compréhension du code. Les noms de nœuds peuvent être quasiment arbitraire mais ils ne doivent pas contenir de virgules, points, parenthèses, deux points ni quelques autres caractères spéciaux.

La syntaxe de l'opération `node` est assez libérale en ce qui concerne l'ordre dans lequel le nom de nœud, le spécificateur `at` et les options doivent apparaître. En fait, on peut même avoir des blocs multiples d'options entre le `node` et le texte entre accolades, ils s'additionnent.

On peut les ranger de manière arbitraire et peut-être que ce qui suit est préférable :

```
\begin{tikzpicture}
  \node[place]      (waiting 1)      at ( 0,2) {};
  \node[place]      (critical 1)     at ( 0,1) {};
  \node[place]      (semaphore)      at ( 0,0) {};
  \node[transition] (leave critical) at ( 1,1) {};
  \node[transition] (enter critical) at (-1,1) {};
\end{tikzpicture}
```

2.8. Placer des nœuds avec le placement relatif

Bien que Hagen désire toujours joindre les nœuds, il souhaite d'abord s'occuper d'un autre problème une fois encore : le placement des nœuds. Bien qu'il aime la syntaxe avec `at`, dans ce cas particulier, il préférerait placer les nœuds « l'un relativement aux autres ». Aussi, Hagen voudrait pouvoir dire que le nœud `critical 1` devrait être sous le nœud `waiting 1`, quelle que soit la place de ce dernier. On peut obtenir cela de plusieurs manières mais la plus jolie, dans le cas de Hage, est l'option `below of` (*en-dessous de*).

```
\begin{tikzpicture}
  \node[place]      (waiting)
                                       {};
  \node[place]      (critical)
                        [below of=waiting]   {};
  \node[place]      (semaphore)
                        [below of=critical] {};
  \node[transition] (leave critical)
                        [right of=critical] {};
  \node[transition] (enter critical)
                        [left of=critical]  {};
\end{tikzpicture}
```

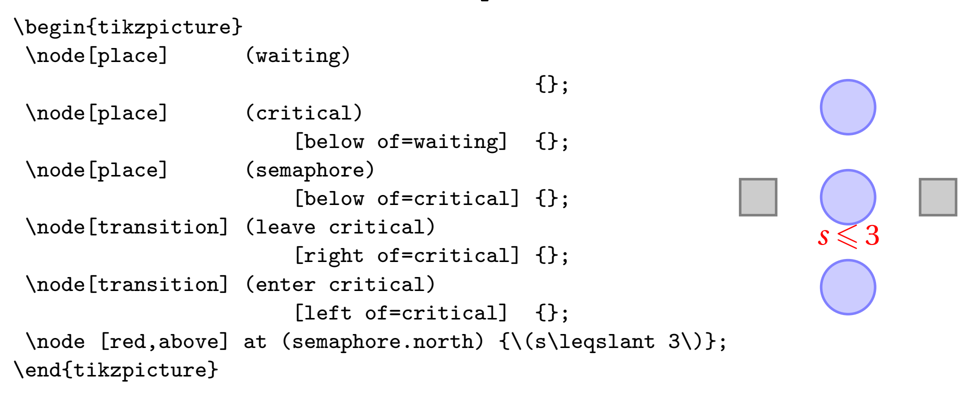
L'option `below of` et ses semblables déterminent la position du nœud de telle sorte qu'il soit placé à la distance `node distance` dans la direction spécifiée. La distance `node distance` est la distance qui sépare les centres des nœuds et pas leurs frontières.

Même si le code ci-dessus a le même effet que le code précédent, Hagen peut le passer à un de ses collègues qui pourra le comprendre rien qu'en le lisant, peut-être même sans avoir vu la figure.

2.9. Ajouter des étiquettes textuelles aux nœuds

Avant de voir comment Hagen peut joindre les nœuds ajoutons la capacité « $s \leqslant 3$ » au nœud du bas. Pour cela deux approches sont possibles :

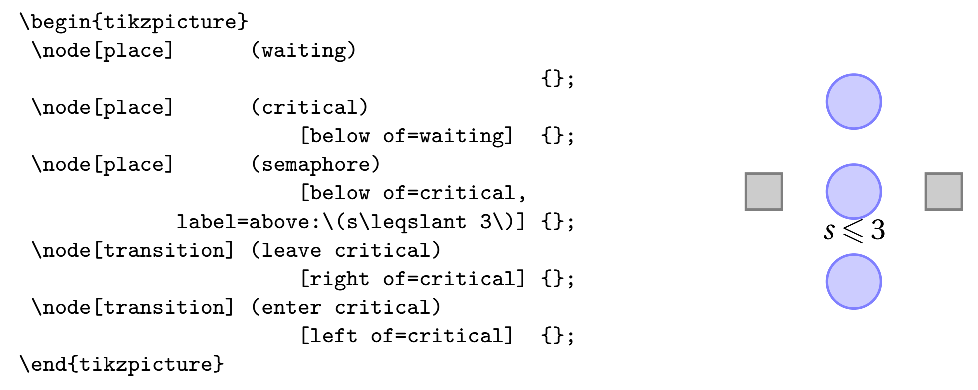
1. Hagen peut se contenter d'ajouter un nouveau nœud au-dessus de l'ancre `north` (*nord*) du nœud `semaphore`.



```
\begin{tikzpicture}
 \node[place]      (waiting)
                                       {};
 \node[place]      (critical)
                      [below of=waiting]  {};
 \node[place]      (semaphore)
                      [below of=critical] {};
 \node[transition] (leave critical)
                      [right of=critical] {};
 \node[transition] (enter critical)
                      [left of=critical]  {};
 \node [red,above] at (semaphore.north) {\(s\leqslant 3\)};
\end{tikzpicture}
```


C'est une approche générale qui « marchera toujours ».

2. Hagen peut utiliser l'option spéciale `label`. Cette option est passée à un nœud et ajoute un *autre* nœud près de celui auquel on a passé l'option. Voici l'idée : quand on construit le nœud `semaphore`, on souhaite indiquer que l'on veut, au-dessus de lui, un autre nœud contenant la capacité. Pour cela, on utilise l'option `label=above:\(s\leqslant 3\)`. Cette option est interprétée comme suit : on veut un nœud au-dessus (*above*) du nœud `semaphore` et ce nœud présentera le texte « $s \leqslant 3$ ». Au lieu de `above` on peut également utiliser, avant le deux-points, des choses comme `below` `left` ou un nombre comme `60`.



```
\begin{tikzpicture}
 \node[place]      (waiting)
                                       {};
 \node[place]      (critical)
                      [below of=waiting]  {};
 \node[place]      (semaphore)
                      [below of=critical,
             label=above:\(s\leqslant 3\)] {};
 \node[transition] (leave critical)
                      [right of=critical] {};
 \node[transition] (enter critical)
                      [left of=critical]  {};
\end{tikzpicture}
```


On peut aussi donner de multiples options `label` ce qui entrainera la

présente de multiples étiquettes.

```
\shorthandoff{:!}%
\tikz
\node [circle,draw,label=60:$60\degres$,
       label=below:$-90\degres$] {mon cercle};
```

Hagen n'est pas totalement satisfait de l'option `label` car l'étiquette n'est pas rouge. Pour obtenir cela il a deux options : premièrement, il peut redéfinir le style `every label` (*chaque étiquette*) ; deuxièmement, il peut ajouter des options au nœud d'étiquette. Ces options sont données à la suite de `label=` et il pourrait écrire `label=[red]above:\(s\leqslant 3\)`. Toutefois, cela ne fonctionne pas complètement car TeX pense que le `]` ferme la liste des options du nœud `semaphore`. Aussi, Hagen doit-il ajouter des accolades et écrire : `label={[red]above:\(s\leqslant 3\)}`. Comme cela semble affreux Hagen décide de redéfinir le style `every label`.

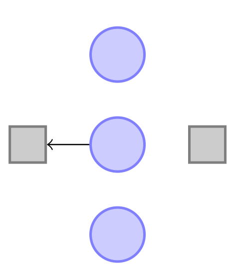
```
\begin{tikzpicture}
 \tikzstyle{every label}=[red]
 \node[place]      (waiting)
                                     {};
 \node[place]      (critical)
                       [below of=waiting]  {};
 \node[place]      (semaphore)
                       [below of=critical,
            label=above:\(s\leqslant 3\)] {};
 \node[transition] (leave critical)
                       [right of=critical] {};
 \node[transition] (enter critical)
                       [left of=critical]  {};
\end{tikzpicture}
```

2.10. Joindre les nœuds

Il est vraiment temps maintenant de joindre les nœuds. Commençons par quelque chose de simple, à savoir la ligne droite entre `enter critical` et `critical`. Nous voulons que cette ligne commence du côté droit de `enter critical` et s'arrête sur le côté gauche de `critical`. Pour cela, nous pouvons utiliser les *ancres* des nœuds. Chaque nœud définit une floppée d'ancres qui sont situées sur sa frontière ou dans son intérieur. Par exemple l'ancre `center` (*centre*) est au

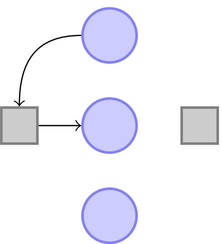
centre du nœud, l'ancre `west` (*ouest*) est à la gauche du nœud et ainsi de suite. Pour accéder à la coordonnée d'un nœud on utilise le nom de ce nœud suivi d'un point lui-même suivi du nom de l'ancre :

```
\begin{tikzpicture}
  \node[place]      (waiting)                          {};
  \node[place]      (critical)       [below of=waiting]  {};
  \node[place]      (semaphore)      [below of=critical] {};
  \node[transition] (leave critical) [right of=critical] {};
  \node[transition] (enter critical) [left of=critical]  {};
  \draw [->] (critical.west) -- (enter critical.east);
\end{tikzpicture}
```



Ensuite, occupons-nous de la courbe joignant `waiting` à `enter critical`. Cela peut être déterminer à l'aide de courbes avec points de contrôle :

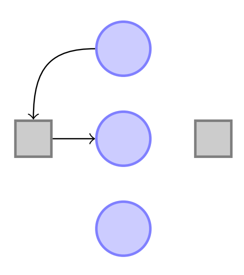
```
\begin{tikzpicture}
  \node[place]      (waiting)                          {};
  \node[place]      (critical)       [below of=waiting]  {};
  \node[place]      (semaphore)      [below of=critical] {};
  \node[transition] (leave critical) [right of=critical] {};
  \node[transition] (enter critical) [left of=critical]  {};
  \draw [->] (enter critical.east) -- (critical.west);
  \draw [->] (waiting.west) .. controls +(left:5mm) and +(up:5mm)
                            .. (enter critical.north);
\end{tikzpicture}
```



Hagen voit désormais comment ajouter toutes les arêtes mais le processus semble étrange et pas très souple. Là encore, le code semble cacher la structure du graphique.

Aussi commençons à améliorer le code des arêtes. Tout d'abord, Hagen peut laisser tomber les ancres [5] :

```
\begin{tikzpicture}
  \node[place]      (waiting)                          {};
  \node[place]      (critical)       [below of=waiting]  {};
  \node[place]      (semaphore)      [below of=critical] {};
  \node[transition] (leave critical) [right of=critical] {};
  \node[transition] (enter critical) [left of=critical]  {};
  \draw [->] (enter critical) -- (critical);
  \draw [->] (waiting) .. controls +(left:8mm) and +(up:8mm)
                       .. (enter critical);
\end{tikzpicture}
```



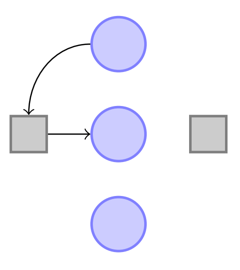
Hagen est un peu surpris de voir que cela fonctionne. Après tout, comme Ti*k*Z sait-il que l'arête joignant `enter critical` à `critical` doit bien commencer sur les frontières ? À chaque fois que Ti*k*Z rencontre un nom complet de nœud comme « coordonnée » il essaie « d'être intelligent » en choisissant l'ancre pour ce nœud. Suivant ce qui viendra après, Ti*k*Z choisira une ancre qui se trouve sur la frontière du nœud sur une droite joignant ce nœud à la prochaine coordonnée ou au prochain point de contrôle. Les règles exactes sont un peu compliquées mais le point choisi sera, en général, correct ; et quand ce ne sera pas le cas, Hagen pourra toujours définir l'ancre voulue à la main.

Hagen voudrait maintenant simplifier un peu l'opération `curve`. Il se fait que cela peut être accompli avec une opération spéciale de chemin : l'opération `to`. Cette opération prend de nombreuses options (on peut même définir les siennes). Une paire d'option est utiles pour Hagen : la paire `in` (*dans*) et `out` (*dehors*). Ces options prennent des angles auxquels

5. Métaphore maritime ! [N.D.T.]

la courbe doit quitter ou atteindre les coordonnées de la source ou de la cible. Sans ces options, c'est une ligne droite qui est tracée :

```
\begin{tikzpicture}
  \node[place]      (waiting)                            {};
  \node[place]      (critical)       [below of=waiting]  {};
  \node[place]      (semaphore)      [below of=critical] {};
  \node[transition] (leave critical) [right of=critical] {};
  \node[transition] (enter critical) [left of=critical]  {};
  \draw [->] (enter critical) to                 (critical);
  \draw [->] (waiting)        to [out=180,in=90] (enter critical);
\end{tikzpicture}
```



L'opération `to` admet encore une autre option, qui convient encore mieux au problème de Hagen : l'option `bend right` (*tournant à droite*). Cette option prend également un angle mais cette valeur ne détermine que l'angle par lequel la courbe tourne vers la droite :

```
\begin{tikzpicture}
  \node[place]      (waiting)              {};
  \node[place]      (critical)
                        [below of=waiting]  {};
  \node[place]      (semaphore)
                        [below of=critical] {};
  \node[transition] (leave critical)
                        [right of=critical] {};
  \node[transition] (enter critical)
                        [left of=critical]  {};
  \draw [->] (enter critical) to
                                 (critical);
  \draw [->] (waiting)        to
               [bend right=45] (enter critical);
  \draw [->] (enter critical) to
                    [bend right=45] (semaphore);
\end{tikzpicture}
```

Il est temps maintenant pour Hagen d'apprendre encore une autre manière de déterminer les arêtes : l'opération de chemin `edge` (*arête*). Cette opération est très semblable à l'opération `to` mais il y a une différence importante : de la même manière qu'un nœud, l'arête créée par l'opération `edge` ne fait pas partie du chemin principal mais n'est

ajoutée que postérieurement. Cela peut ne pas sembler très important mais a quelques conséquences agréables. Par exemple, chaque arête peut avoir ses propres pointes de flèche et sa propre couleur et ainsi de suite et, de plus, on peut donner toutes les arêtes sur le même chemin. Cela permet à Hagen d'écrire ce qui suit :

```
\begin{tikzpicture}
  \node[place]      (waiting)              {};
  \node[place]      (critical)
                         [below of=waiting]  {};
  \node[place]      (semaphore)
                         [below of=critical] {};
  \node[transition] (leave critical)
                         [right of=critical] {};
  \node[transition] (enter critical)
                         [left of=critical]  {}
    edge [->]               (critical)
    edge [<-,bend left=45]  (waiting)
    edge [->,bend right=45] (semaphore);
\end{tikzpicture}
```

Chaque edge fait qu'un nouveau chemin est construit, consistant en un to entre le nœud enter critical et le nœud qui suit la commande edge.

Pour la touche finale on introduit deux styles pre et post et on utilise l'option bend angle=45 qui définit les deux angles de courbure d'un coup :

```
% Styles place et transition comme précédemment
\tikzstyle{pre}=[<-,shorten <=1pt,>=stealth',semithick]
\tikzstyle{post}=[->,shorten >=1pt,>=stealth',semithick]
\begin{tikzpicture}[bend angle=45]
  \node[place]      (waiting)             {};
  \node[place]      (critical)
                        [below of=waiting]  {};
  \node[place]      (semaphore)
                        [below of=critical] {};

  \node[transition] (leave critical)
                        [right of=critical] {}
    edge [pre]             (critical)
    edge [post,bend right] (waiting)
    edge [pre, bend left]  (semaphore);
  \node[transition] (enter critical)
                        [left of=critical]  {}
    edge [post]            (critical)
    edge [pre, bend left]  (waiting)
    edge [post,bend right] (semaphore);
\end{tikzpicture}
```

2.11. Ajouter des étiquettes près des lignes

Ce que fait Hagen ensuite c'est ajouter « 2 » près des arcs. Pour cela Hagen peut utiliser le placement automatique de nœud de Ti*k*Z : si l'option `auto` est ajoutée, Ti*k*Z placera les nœuds de telle sorte qu'ils ne soient pas sur la courbe mais à côté d'elle. En ajoutant `swap` on obtiendra la position symétrique de l'étiquette par rapport à la courbe. Voici un exemple général :

```
\begin{tikzpicture}[auto,bend right]
  \node (a) at (0:1) {$0\degres$};
  \node (b) at (120:1) {$120\degres$};
  \node (c) at (240:1) {$240\degres$};

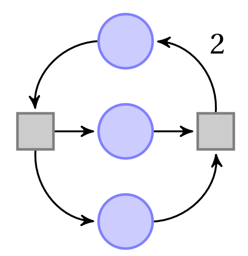
  \draw (a) to node {1} node [swap] {$1'$} (b)
        (b) to node {2} node [swap] {$2'$} (c)
        (c) to node {3} node [swap] {$3'$} (a);
\end{tikzpicture}
```


Que se passe-t-il ici ? Les nœuds sont donnés à l'intérieur de l'opération `to` d'une certaine manière ! Quand on fait cela, le nœud est placé au milieu de la courbe, ou droite, créée par l'opération `to`. L'option `auto` fait alors que le nœud soit déplacé de telle sorte qu'il ne soit pas sur la courbe mais à côté. Dans cet exemple on fournit même deux nœuds pour chaque opération `to`.

Pour Hagen cette option `auto` n'est pas vraiment nécessaire puisque les deux étiquettes « 2 » pourraient être facilement placées « à la main ». Toutefois, dans une figure complexe contenant de nombreuses arêtes le placement automatique peut se révéler une bénédiction.

```
% Styles comme précédemment
\begin{tikzpicture}[bend angle=45,scale=4]
  \node[place]      (waiting)                           {};
  \node[place]      (critical)       [below of=waiting]  {};
  \node[place]      (semaphore)      [below of=critical] {};

  \node[transition] (leave critical) [right of=critical] {}
    edge [pre]                                 (critical)
    edge [post,bend right] node[auto,swap] {2} (waiting)
    edge [pre, bend left]                      (semaphore);
  \node[transition] (enter critical) [left of=critical]  {}
    edge [post]                                (critical)
    edge [pre, bend left]                      (waiting)
    edge [post,bend right]                     (semaphore);
\end{tikzpicture}
```



2.12. Ajout de la ligne serpentante et du texte multiligne

Grâce au mécanisme des nœuds Hagen peut facilement créer les deux réseaux de Petri. Il ne sait cependant pas encore comment créer la ligne serpentante entre les deux réseaux.

Pour cela il peut utiliser un *serpent* (snake). Les serpents sont ainsi nommés car la forme la plus élémentaire d'un serpent ressemble exactement à un serpent. Toutefois n'importe quel motif répétitif peut être utilisé comme serpent, motif tel que des bosses ou des dents de scie ou même des choses bien plus compliquées.

Pour tracer le serpent il suffit à Hagen de placer l'option `snake=snake` sur le chemin. Cela fait que toutes les lignes droites du chemin sont remplacées par des serpents. On peut également utiliser des serpents que pour certaines parties du chemin mais Hagen n'aura pas besoin de cette possibilité.

```
\begin{tikzpicture}
  \draw [->,snake=snake] (0,0) -- (2,0);
\end{tikzpicture}
```

Bon, ça ne semble pas tout à fait correct pour l'instant. Le problème est que le serpent se trouve finir exactement là où la flèche commence. Heureusement une option peut aider dans ce cas. De plus le serpent devrait être un tout petit peu plus étroit, ce que l'on peut régler avec encore d'autres options.

```
\begin{tikzpicture}
  \draw [->,snake=snake,
         segment amplitude=.4mm,
         segment length=2mm,
         line after snake=1mm] (0,0) -- (3,0);
\end{tikzpicture}
```

Maintenant Hagen doit ajouter le texte au-dessus du serpent. Ce texte pose un petit problème car il est multiligne. Pour saisir un tel texte Hagen doit définir sa largeur et doit préciser qu'il devrait être centré.

```
\begin{tikzpicture}
  \draw [->,snake=snake,
         segment amplitude=.4mm,
         segment length=2mm,
         line after snake=1mm] (0,0) -- (3,0)
    node [above,text width=3cm,text centered,midway]
    {
      remplacement de la
      \textcolor{red}{capacité} par
      \textcolor{red}{deux places}
    };
\end{tikzpicture}
```

remplacement de la capacité par deux places

2.13. Utiliser des couches : les rectangles d'arrière-plan

Hagen doit encore ajouter les rectangles d'arrière-plan. C'est un peu pénible : Hagen voudrait tracer ces rectangles *après* que les réseaux de Petri ont été finis. La raison en est que ce n'est qu'à ce moment qu'il pourra faire référence aux coordonnées qui déterminent les coins du rectangle. Si Hagen veut dessiner les rectangles en premier alors il a besoin de la taille exacte du réseau de Petri — ce qu'il n'a pas.

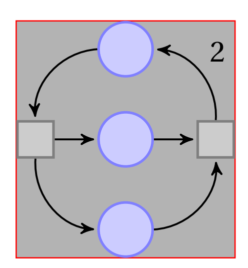
La solution est d'utiliser des *couches* (layer). Quand la bibliothèque d'arrière-plans (*background*) est chargée, Hagen peut placer quelques morceaux de la figure dans un environnement `{pgfonlayer}`. Alors ces morceaux de la figure deviennent des parties de la couche qui est passée en argument de l'environnement. À la fin de l'environnement `{tikzpicture}` les couches sont placées les unes au-dessus des autres, en commençant par la couche d'arrière-plan. Ce qui fait que tout ce qui est dessiné sur la couche de fond est sous le texte principal.

```
% Styles comme précédemment
\begin{tikzpicture}[bend angle=45,scale=4]
  \node[place]      (waiting)                           {};
  \node[place]      (critical)       [below of=waiting]  {};
  \node[place]      (semaphore)      [below of=critical] {};

  \node[transition] (leave critical) [right of=critical] {}
    edge [pre]                                  (critical)
    edge [post,bend right] node[auto,swap] {2} (waiting)
    edge [pre, bend left]                       (semaphore);
  \node[transition] (enter critical) [left of=critical]  {}
    edge [post]                                 (critical)
    edge [pre, bend left]                       (waiting)
    edge [post,bend right]                      (semaphore);

  \begin{pgfonlayer}{background}
    \filldraw [fill=black!30,draw=red]
```

```
                (semaphore.south -| enter critical.west)
      rectangle (waiting.north   -| leave critical.east);
  \end{pgfonlayer}
\end{tikzpicture}
```



2.14. Le code complet

Hagen a maintenant enfin tout rassemblé. Et ce n'est que maintenant qu'il apprend qu'il existe déjà une bibliothèque dédiée aux réseaux de Petri ! Il se trouve que cette bibliothèque fournit essentiellement les mêmes définitions que celles de Hagen. Par exemple, elle définit un style `place` d'une manière similaire à ce qu'a fait Hagen. Ajuster le code pour qu'il fasse appel à la bibliothèque permet de le raccourcir un peu comme ce qui suit le montre.

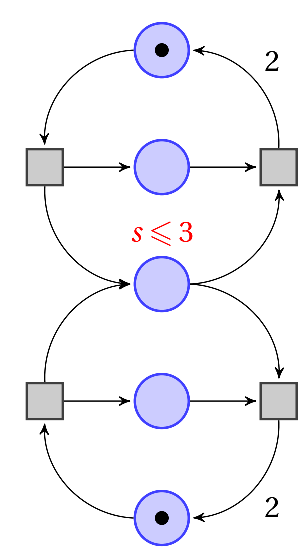
D'abord, Hagen a besoin de moins de définitions de style mais il doit toujours définir les couleurs des places et des transitions.

```
{\shorthandoff{:!} % pour les caractères actifs de frenchb
\begin{tikzpicture}
     [node distance=1.3cm,>=stealth',bend angle=45,auto]
\tikzstyle{every place}=
     [minimum size=6mm,thick,draw=blue!75,fill=blue!20]
\tikzstyle{every transition}=
     [thick,draw=black!75,fill=black!20]
\tikzstyle{red place}=
     [place,draw=red!75,fill=red!20]
\tikzstyle{every label}=
     [red]
```

Et maintenant, les réseaux :

```
\begin{scope}
 \node [place,tokens=1] (w1)                    {};
 \node [place]          (c1)  [below of=w1]      {};
 \node [place]          (s)
       [below of=c1,label=above:$s\leqslant 3$] {};
 \node [place]          (c2)  [below of=s]       {};
 \node [place,tokens=1] (w2)  [below of=c2]      {};
```

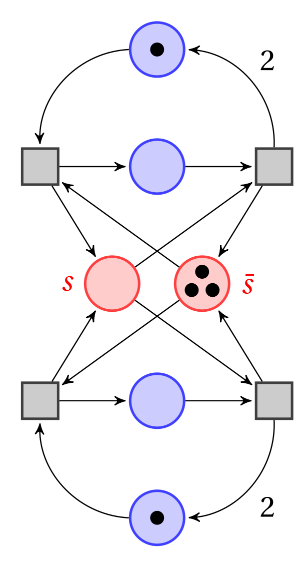
```
 \node [transition] (e1) [left of=c1] {}
   edge [pre,bend left]                  (w1)
   edge [post,bend right]                (s)
   edge [post]                           (c1);
 \node [transition] (e2) [left of=c2] {}
   edge [pre,bend right]                 (w2)
   edge [post,bend left]                 (s)
   edge [post]                           (c2);
 \node [transition] (l1) [right of=c1] {}
   edge [pre]                            (c1)
   edge [pre,bend left]                  (s)
   edge [post,bend right] node[swap] {2} (w1);
 \node [transition] (l2) [right of=c2] {}
   edge [pre]                            (c2)
   edge [pre,bend right]                 (s)
   edge [post,bend left]  node {2}       (w2);
\end{scope}
```




```
\begin{scope}[xshift=6cm]
  \node [place,tokens=1] (w1')                            {};
  \node [place]          (c1') [below of=w1']             {};
  \node [red place]      (s1') [below of=c1',xshift=-5mm]
          [label=left:$s$]                                {};
  \node [red place,tokens=3]
                         (s2') [below of=c1',xshift=5mm]
          [label=right:$\bar s$]                          {};
  \node [place]          (c2') [below of=s1',xshift=5mm]  {};
  \node [place,tokens=1] (w2') [below of=c2']             {};

  \node [transition] (e1') [left of=c1'] {}
    edge [pre,bend left]                  (w1')
    edge [post]                           (s1')
```

```
    edge [pre]                          (s2’)
    edge [post]                         (c1’);
  \node [transition] (e2’) [left of=c2’] {}
    edge [pre,bend right]               (w2’)
    edge [post]                         (s1’)
    edge [pre]                          (s2’)
    edge [post]                         (c2’);
  \node [transition] (l1’) [right of=c1’] {}
    edge [pre]                          (c1’)
    edge [pre]                          (s1’)
    edge [post]                         (s2’)
    edge [post,bend right] node[swap] {2} (w1’);
  \node [transition] (l2’) [right of=c2’] {}
    edge [pre]                          (c2’)
    edge [pre]                          (s1’)
    edge [post]                         (s2’)
    edge [post,bend left]  node {2}     (w2’);
\end{scope}
```



Le code pour l’arrière-plan et le serpent est le suivant :

```
\draw [-to,thick,snake=snake,segment amplitude=.4mm,
       segment length=2mm,line after snake=1mm]
      ([xshift=5mm]s -| l1) -- ([xshift=-5mm]s1’ -| e1’)
 node [above=1mm,midway,text width=3cm,text centered]
      {remplacement de la \textcolor{red}{capacité}
       par \textcolor{red}{deux places}};

 \begin{pgfonlayer}{background}
   \filldraw [line width=4mm,join=round,black!10]
     (w1.north  -| l1.east)  rectangle (w2.south  -| e1.west)
     (w1’.north -| l1’.east) rectangle (w2’.south -| e1’.west);
 \end{pgfonlayer}
```

```
\end{tikzpicture}
} % pour le \shorthandoff
```

3. Conseils à propos des graphiques

Cette section ne concerne pas PGF ou Ti*k*Z mais donne quelques conseils sur la création de graphiques pour des présentations, des articles ou des livres scientifiques.

Les conseils de cette section proviennent de sources différentes. Un grand nombre d'entre eux ressortissent simplement à ce que je voudrais appeler le « bon sens », d'autres reflètent mon expérience personnelle (toutefois, je l'espère, pas mes préférences personnelles), d'autres encore sont issus de livres (la bibliographie est toujours absente, j'en suis désolé) sur la création de graphique et la typographie. Ce qui m'a le plus influencé ce sont les livres brillants d'Edward Tufte. Bien que je ne sois pas d'accord avec tout ce qu'on lit dans ces livres, je trouve que de nombreux arguments de Tufte sont si convaincants que j'ai décidé de les reprendre dans les conseils qui suivent.

3.1. Faut-il suivre ces conseils ?

La première question que l'on devrait se poser lorsque quelqu'un donne un tas de conseils est : dois-je vraiment suivre ces conseils ? C'est une question importante parce qu'il y a de bonnes raisons de ne pas suivre des conseils généraux.

— La personne qui a écrit ces conseils pourrait avoir en vue un autre objectif que le vôtre. Par exemple, un conseil pourrait être : « Utilisez la couleur rouge pour mettre en relief. » Ce conseil est sensé s'il s'agit, par exemple, d'une présentation faite au rétroprojecteur mais la « couleur » rouge a l'effet *opposé* à la mise en relief lorsqu'on imprime avec une imprimante noir et blanc.

Les conseils sont presque toujours rédigés en vue d'une situation particulière. Si l'on n'est pas dans cette situation, suivre ce conseil peut faire plus de mal que de bien.

— La règle de base en typographie est : « On peut contourner une règle tant que l'on *sait* que l'on contourne une règle. » Cette règle s'applique aussi aux graphiques. Dite autrement, la règle de base est : « Les seules erreurs en typographie sont les choses faites par ignorance. »

Lorsque vous connaissez une règle et que vous déterminez que la contourner permet d'obtenir l'effet voulu, contournez-la.

— Certains conseils sont simplement *faux* mais tout le monde les suit par tradition ou est forcé de le faire. Mon exemple favori est la directive qu'une société de conception de logiciels pour laquelle j'ai travaillé avait imposé dans un grand projet : tous les programmeurs doivent déclarer les paramètres des fonctions selon *l'ordre croissant de leur taille*. Ainsi les paramètres d'un octet devaient venir les premiers, puis ceux de deux octets, et ainsi de suite.

Cette directive est un non-sens total.

3.2. Estimer le temps nécessaire à la création de graphiques

Lorsque l'on crée un article avec de nombreux graphiques, le temps nécessaire à créer ces graphiques devient un facteur important. Combien de temps devrait-on prévoir pour la création des graphiques ?

En général, il faut compter qu'un graphique demande autant de temps qu'en demande un texte du même encombrement. Par exemple, lorsque j'écris un article, j'ai besoin d'une heure par page pour le premier jet. Ensuite, j'ai besoin de deux à quatre heures par page pour les relectures et corrections. Aussi je compte environ une demie heure pour créer un *premier jet* d'un graphique d'une demie page. Je m'attends à avoir besoin plus tard d'une à deux heures supplémentaires pour produire le graphique final.

Dans beaucoup de publications, même des revues de qualité, les auteurs et éditeurs ont visiblement investi beaucoup de temps sur le texte mais semblent avoir passé environ cinq minutes pour créer tous les graphiques. Les graphiques semblent souvent avoir été ajoutés après coup ou ressemblent à une copie d'écran du logiciel de statistique qu'utilise l'auteur. Les graphiques produits par de tels outils sans réflexion graphique ni adaptation à la maquette de l'article en cours sont en général médiocres.

Créer des graphiques instructifs qui aident le lecteur et se marient correctement avec le texte principal est un processus long et difficile.

— Traitez les graphiques comme des éléments importants de vos articles. Ils méritent autant de temps et d'énergie que le texte.

— On peut avancer que la création de graphiques mérite *même plus* de temps que l'écriture du texte principal parce qu'on sera plus attentif

aux graphiques et qu'on les regarda les premiers.

— Prévoyez autant de temps pour la création et la correction d'un graphique que vous n'en prévoyez pour un texte de la même taille.

— Les graphiques difficiles contenant beaucoup d'informations peuvent même demander plus de temps.

— Des graphiques très simple demanderont moins de temps mais, très probablement, vous ne voulez pas de « graphiques très simples » dans votre article, de toute façon ; de même que vous ne voudriez pas de « texte très simple » de la même taille.

3.3. Processus de création de graphique

Lorsque l'on écrit un article (scientifique), on suit probablement le modèle suivant : on a quelques résultats ou idées qu'on voudrait publier. La création de l'article commencera d'habitude par un plan sommaire. Puis, on remplira les différentes parties avec du texte pour créer un premier brouillon. Ce brouillon sera revu de manière répétée jusqu'à ce que, souvent après d'importantes corrections, un article fini en émerge. Il n'y a en général pas une seule phrase du brouillon d'un bon article de revue qui ait survécu sans modification.

La création de graphiques suit le même modèle :

— Décider ce que le graphique doit communiquer. En faire une décision consciente, c'est-à-dire répondre à la question : « Qu'est-ce que ce graphique est sensé dire au lecteur ? »

— Créer un croquis, c'est-à-dire une forme générale et grossière du graphique, contenant les éléments les plus importants. Souvent il est utile de le faire avec papier et crayon.

— Placer les détails plus précis pour créer le premier brouillon.

— Reprendre ce graphique de manière répétée en même temps que le reste de l'article.

3.4. Lier le graphique avec le texte principal

On peut placer les graphiques à différents endroits dans un article. Ou bien on les place « dans le texte », c.-à-d. quelque part au milieu d'un paragraphe ou entre deux paragraphes ou bien on les place à part, « en hors-texte ». Comme les imprimeurs (et les gens en général) aiment bien les pages pleines (autant pour des raisons esthétiques qu'économiques) les figures hors-texte peuvent traditionnellement être placées sur des

pages très éloignées du texte principal dans lequel on y fait référence. LaTeX et TeX tendent à encourager ce rejet des graphiques pour des raisons techniques.

Quand une figure est dans le texte, elle est plus ou moins automatiquement liée au texte principal dans le sens que ses annotations seront implicitement expliquées par le texte environnant. Ainsi le texte principal explicitera en général le pourquoi de la figure et ce qu'elle montre.

Il en est tout à fait autrement pour une figure hors-texte qui sera vue lorsque le texte auquel elle est liée soit n'aura pas encore été lu soit aura été lu bien longtemps avant. Pour cela on devrait, en créant un figure hors-texte, suivre les conseils que voici :

— Les figures hors-texte devraient avoir une légende qui les rende « compréhensibles par elles-mêmes ».

Supposons, par exemple, qu'une figure montre un exemple de différentes étapes de l'algorithme de tri rapide (*quicksort*). La légende de la figure devrait, au minimum, informer le lecteur de ce que « La figure montre les différentes étapes de l'algorithme de tri *quicksort* présenté en page X » et non pas juste « Algorithme *quicksort* ».

— Une bonne légende donne autant de contexte que possible. Par exemple, on pourrait écrire « La figure montre les différentes étapes de l'algorithme de tri *quicksort* présenté en page X. Dans la première ligne, on a pris l'élément 5 pour pivot. Cela entraîne... » Bien que cette information puisse être donnée aussi dans le texte principal, en la plaçant dans la légende on garantira que le contexte est donné. N'ayez pas peur des légendes de 5 lignes de long. (Votre éditeur vous haïra peut-être pour ça. Envisagez de le haïr en retour.)

— On renverra à la figure, depuis le texte principal, avec quelque chose comme « Pour un exemple de *quicksort* en action, voir la figure 2.1, page Y. »

— La plupart des ouvrages sur la typographie et des marches d'éditeur recommandent de ne pas utiliser les abréviations comme « fig. 2.1 » mais d'écrire, au long, « figure 2.1 ».

L'argument principal contre les abréviations est que « le point est trop précieux pour le gaspiller dans une abréviation ». L'idée est que le point amène le lecteur à penser que la phrase s'arrête après « fig » et qu'il doit « revenir consciemment en arrière » pour se rendre compte que la phrase ne se finissait pas là après tout.

L’argument en faveur des abréviations est qu’elles économisent de la place.

Personnellement je ne suis convaincu par aucun de ces arguments. D’un côté, je n’ai jamais vu de preuve sérieuse que les abréviations ralentissent la lecture. D’un autre, abréger tous les « figure » en « fig. » a peu de chance d’économiser ne fusse qu’une seule ligne dans la plupart des documents.

J’évite les abréviations.

3.5. Cohérence du texte et des figures

Peut-être que « l’erreur » la plus fréquemment faite quand on crée des graphiques (souvenez-vous qu’une « erreur » de conception est toujours juste de « l’ignorance ») est d’avoir une discordance entre l’aspect des graphiques et celui du texte.

Il est fréquent que les auteurs utilisent différents logiciels pour créer les graphiques de leur article. Un auteur pourra produire quelques courbes à l’aide de GNUPLOT, un diagramme avec XFIG et incorporer une image `.eps` produite par un co-auteur utilisant un logiciel inconnu. Tous ces graphiques auront, le plus vraisemblablement, des traits d’épaisseurs différentes, des polices différentes, des tailles différentes. De plus, les auteurs utilisent souvent, en important les graphiques, des options comme `[height=5cm]` pour les réduire à une « taille sympa ».

Si la même approche était suivie dans l’écriture du texte principal, chaque partie serait écrite avec une police et dans un corps différents. Dans quelques parties les théorèmes seraient soulignés, dans d’autres il seraient en capitales, dans d’autres encore écrits en rouge. De plus, les marges changeraient d’une page à l’autre.

Les lecteurs et les éditeurs ne toléreraient pas qu’un texte soit écrit de cette façon [6] mais ils tolèrent ce genre de choses pour les graphiques.

Pour assurer une cohérence entre les graphiques et le texte, suivez attentivement ces conseils :

— Ne réduisez ni n’agrandissez les graphiques.

Cela signifie que, lorsque l’on crée une figure avec un programme externe, on doit la créer « à la bonne taille ».

— Utilisez la même police dans la figure et le corps de texte.

6. Il me semble que l’auteur fait preuve ici d’un optimisme exagéré. [N.D.T.]

— Utilisez la même épaisseur de trait dans le texte et les figures.

La « largeur de ligne » d'un texte normal est l'épaisseur du fût d'une lettre comme le T. Pour les polices CM de TEX, cela signifie habituellement 0,4 pt. Toutefois quelques revues n'accepteront pas de figures dont l'épaisseur du trait normal est inférieure à un demi point.

— Lorsque vous utilisez des couleurs, faites-le en suivant un code de couleur cohérent tant dans le texte que dans les figures. Par exemple, si le rouge est sensé attirer l'attention du lecteur sur quelque chose dans le texte principal, utilisez-le dans les figures pour les parties importantes. Si le bleu est utilisé pour les éléments structurels tels que les titres de sections, utilisez aussi le bleu pour les éléments structurels des figures.

Toutefois les figures peuvent avoir un code de couleur intrinsèque et logique. Par exemple, quelle que soit la couleur utilisée normalement les lecteurs penseront en général, par exemple, le vert comme « positif, passez, d'accord » et le rouge comme « attention, alerte, action ».

Assurer la cohérence en utilisant différents logiciels de production de graphiques est quasi impossible. On devra donc envisager de n'en utiliser qu'un seul.

3.6. Annotations dans les figures

Presque toutes les figures contiennent des annotations, c.-à-d. des bouts de texte expliquant des parties du graphiques. Lorsque vous placez ces annotations, suivez ces conseils :

— Soyez cohérent en plaçant les annotations et cela sur deux plans : premièrement, soyez cohérent avec le texte principal, c.-à-d. utilisez la même police pour les annotations que pour le texte principal ; deuxièmement, assurez la cohérence entre annotations, c.-à-d. que si vous présentez certaines annotations d'une manière particulière, vous devriez présenter de la même manière toutes les annotations.

— Non seulement vous devriez utilisez les mêmes polices pour le texte et pour les graphiques mais en plus les mêmes notations. Par exemple, si vous écrivez 1/2 dans votre texte principal, utilisez aussi « 1/2 » pour les annotations de vos figures et pas « 0,5 ». π est « π » et non « 3,141 ». Enfin $e^{-i\pi}$ est « $e^{-i\pi}$ », et pas « -1 » et encore moins « -1 ».

— Les annotations devraient être lisibles. Non seulement elles devraient avoir une taille raisonnable mais de plus elles ne devraient pas être recouvertes par des lignes ou d'autres textes. Cela s'applique aussi

aux lignes et aux textes placés *derrière* les annotations.

— Les annotations devraient être « en place ». À chaque fois qu'il y a assez de place, l'annotation devrait être placée près de ce qu'elle marque. On ajoutera une ligne (discrète) entre l'annotation et l'objet auquel elle se rapporte seulement si nécessaire. Essayez d'éviter les annotations qui ne font que référence à des explications situées dans des légendes extérieures. Le lecteur doit alors sauter d'avant en arrière entre les explications et l'objet décrit.

— Pensez à atténuer les annotations « accessoires » en les colorant en gris par exemple. Cela permettra de centrer l'attention sur la figure elle-même.

3.7. Courbes et diagrammes

L'espèce la plus fréquente de graphiques, spécialement dans les articles scientifiques, est celle des *courbes et diagrammes*. Elle présente de nombreuses variétés comprenant les courbes simples, les courbes paramétriques, les surfaces, les histogrammes, les diagrammes en camembert, et bien d'autres encore.

Malheureusement, il est de notoriété publique qu'il est difficile d'obtenir ces graphiques correctement. En partie, on peut blâmer les réglages par défaut des logiciels comme `gnuplot` ou Excel car ces logiciels rendent très facile la création de mauvais diagrammes ou de mauvaises courbes.

La première question à se poser lorsque l'on crée une courbe est :

— Y-a-t'il assez de données pour mériter une courbe ?

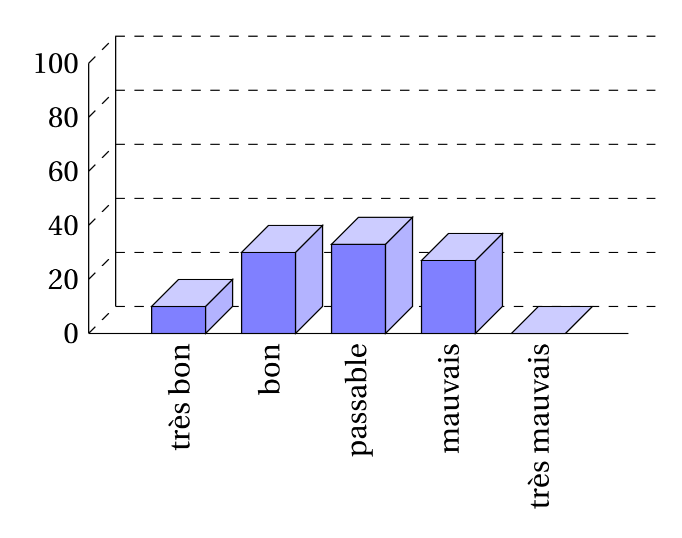
Si la réponse est « pas vraiment », utilisez une table.

Une situation typique où on n'a pas besoin d'un diagramme est lorsque des gens présentent quelques nombres avec un diagramme à barres. Voici un exemple vécu : à la fin d'un séminaire, un intervenant demande leur avis aux participants. D'après cette petite enquête, trois participants considéraient le séminaire comme « très bon », neuf comme « bon », dix comme « passable », huit comme « mauvais » et personne ne qualifiait le séminaire de « très mauvais ».

Une façon simple de résumer l'information est la table suivante :

Classement	*Nombre de participants (sur 50) pour ce classement*	*Pourcentage*
« très bon »	3	6 %
« bon »	9	18 %
« passable »	10	20 %
« mauvais »	8	16 %
« très mauvais »	0	0 %
sans opinion	20	40 %

Ce que fit l'intervenant fut de présenter les données à l'aide d'un diagramme à barres en 3 dimensions. Cela ressemblait à ceci :



La table et le « diagramme » ont à peu près la même taille. Si vous pensez d'abord que « le diagramme semble plus beau que la table », essayez de répondre aux questions suivantes d'après les informations données par la table ou par le graphique :

1. Combien y-avait-il de participants ?
2. Combien de participants ont répondu à l'enquête ?
3. Parmi tous les participants, quel pourcentage a répondu à l'enquête ?
4. Combien de participants ont coché « très bon » ?
5. Parmi tous les participants, quel pourcentage a coché « très bon » ?
6. Y eut-il plus d'un quart des participants pour cocher « mauvais » ou « très mauvais » ?

7. Quel pourcentage de participants a rendu l'enquête en ayant coché « très bon » ?

Malheureusement le diagramme ne permet de répondre à *aucune de ces questions*. La table répond à toutes directement, sauf à la dernière. Essentiellement la densité d'information du graphique est très proche de zéro. La table a une bien plus grande densité d'information en dépit du fait qu'elle utilise beaucoup d'espace blanc pour présenter quelques nombres.

Voici la liste de ce qui n'allait pas dans le diagramme 3D :

— L'ensemble du graphique est dominé par d'irritants traits en arrière plan.

— On ne comprend pas clairement ce que signifient les nombres de gauche ; vraisemblablement des pourcentages mais ce pourraient être aussi des nombres absolus de participants.

— Les annotations du bas ont subi une rotation ce qui les rend difficiles à lire.

(Dans la présentation que j'ai vue, le texte était rendu dans une très basse résolution d'environ 10 fois 6 pixels par lettre avec de mauvaises approches, rendant ce texte presque impossible à lire.)

— La troisième dimension ajoute de la complexité au graphique sans apporter d'information.

— La présentation en trois dimension rend très difficile l'estimation de la hauteur des barres. Considérons la barre de « mauvais ». Représente-t-elle plus ou moins de 20 ? Alors que le devant de la barre est sous la ligne de 20, l'arrière (qui compte) est au-dessus.

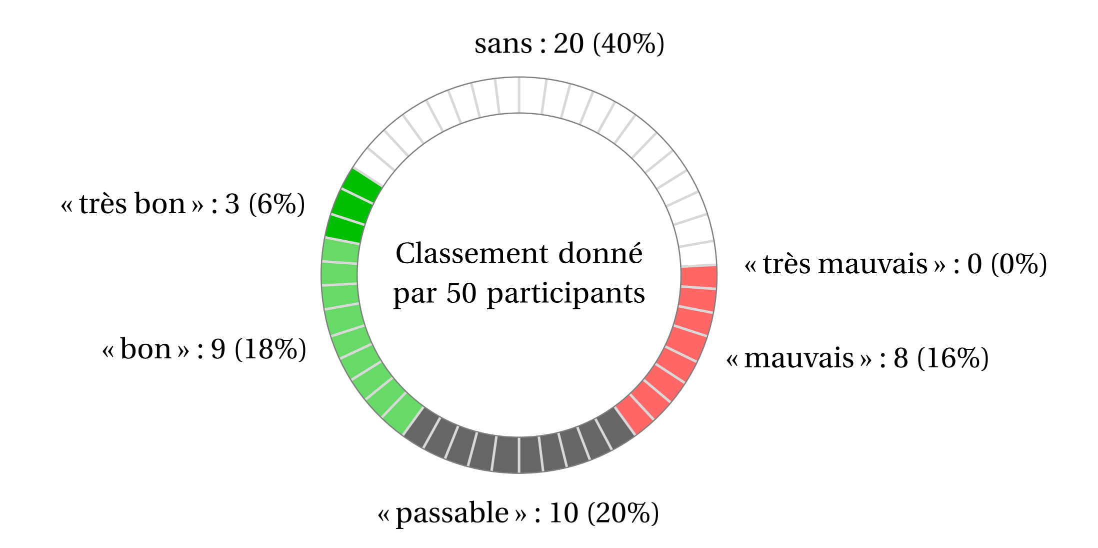
— Il est impossible de dire quels sont les nombres représentés par les barres. Aussi les barres cachent-elles gratuitement l'information qu'elles devaient présenter.

— Que représente la somme des hauteurs des barres : 100 % ou 60 % ?

— La barre de « très mauvais » représente-t-elle 0 ou 1 ?

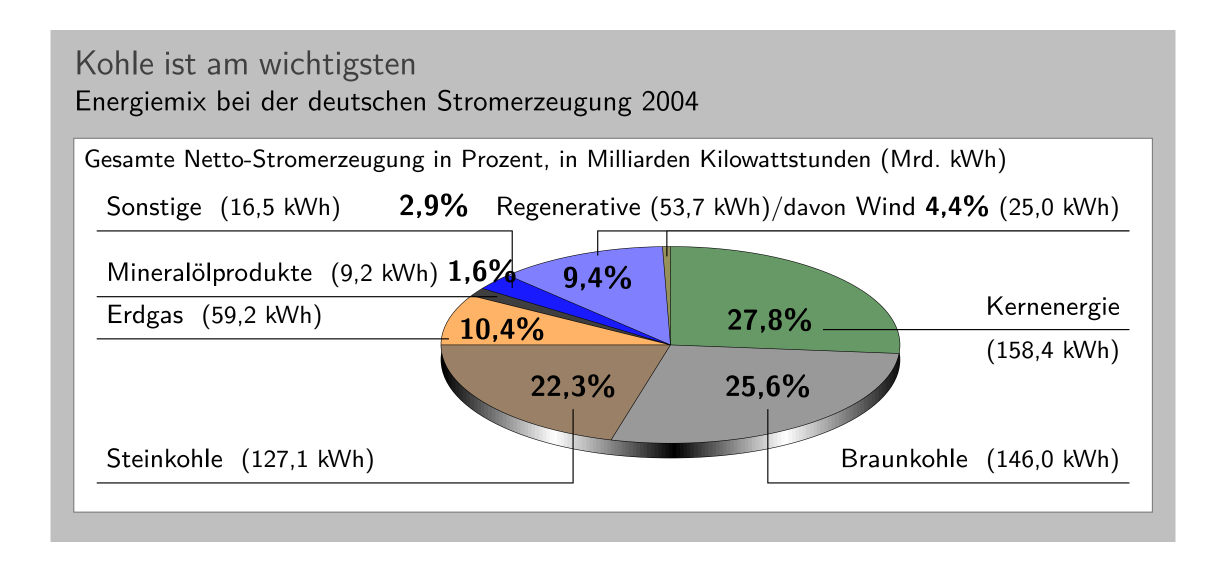
— Pourquoi les barres sont-elles bleues ?

On pourrait avancer que dans cet exemple les nombres exacts ne sont pas importants pour le diagramme. L'important est le « message » qui est qu'il y a plus de « très bon » et de « bon » que de « mauvais » et « très mauvais ». Toutefois, s'il s'agit de faire passer ce message autant utiliser une phrase qui le dit ou un graphique qui le présente plus clairement :



Le graphique ci-dessus a à peu près la même densité d'information que la table (il a à peu près la même taille et présente à peu près les mêmes nombres). De plus, on peut « voir » directement qu'il y a plus de bons classements que de mauvais. On peut aussi « voir » que le nombre de personne qui n'ont pas donné de classement n'est pas négligeable, ce qui est fréquent pour une telle enquête.

Un diagramme n'est pas toujours une bonne idée. Regardons un exemple que je redessine d'après un diagramme circulaire de *Die Zeit* du 4 juin 2005 :



Ce graphique a été redessiné avec Ti*k*Z, mais l'original lui ressemble beaucoup.

À première vue le graphique semble « beau et instructif » mais il y a beaucoup de choses qui ne vont pas :

— Le diagramme est en trois dimensions. Toutefois, les ombres n'apportent rien en terme d'information et, au mieux, elles perturbent.

— Dans un diagramme circulaire en 3D, les tailles relatives sont très déformées. Par exemple, l'aire de la zone grise du « Braunkohle » est plus grande que celle de la zone verte du « Kernenergie » *en dépit du fait que le pourcentage de Braukohle est plus petit que celui de Kernenergie.*

— La distortion due à l'effet 3D est encore pire pour les petites surfaces. L'aire de « Regenerative » est un peu plus grande que celle de « Erdgas ». L'aire de « Wind » est légèrement plus petite que celle de « Mineralölprodukte » *alors même que le pourcentage du Wind est presque trois fois plus grand que celui de Mineralölprodukte.*

Dans ce dernier cas, les différences dans les tailles ne sont dues qu'en partie à la distortion. Le (ou les) dessinateur(s) du graphique originel ont fait la part du « Wind » trop petit même en tenant compte de la distortion. (Comparez simplement la taille de « Wind » et celle de « Regenerative » en général.)

— D'après sa légende, ce diagramme est sensé nous dire que le charbon est la plus importante source d'énergie en Allemagne en 2004. En laissant de côté les distortions dues à l'effet superflu et trompeur de la 3D, cela prend un temps certain pour capter ce message.

Le charbon, comme source d'énergie, est partagée en deux secteurs : un pour le « Steinkohle » et un pour le « Braukohle » (deux sortes différentes de charbon). Lorsqu'on les ajoute, on voit que toute la partie inférieure du diagramme est consacrée au charbon.

Les deux aires des différentes sortes de charbon ne sont pas du tout reliées visuellement. Plutôt, deux couleurs différentes sont utilisées, les annotations sont sur des côtés différents du graphique. Par comparaison, « Regenerative » et « Wind » sont très intimement reliés.

— Le code de couleur du graphique ne suit aucune structure logique. Pourquoi l'énergie nucléaire en vert ? L'énergie renouvelable est en bleu pâle, les « autres sources » en bleu. On a presque l'impression d'une plaisanterie quand on voit que la surface pour « Braunkohle » (qui se traduit littéralement par « charbon marron ») est en gris pierre alors que

la surface pour « Steinkohle » (qu'on traduit littéralement par « charbon de pierre ») est marron.

— La surface qui a la couleur la plus claire est utilisée pour « Erdgas ». Cette surface est celle qui ressort le plus à cause de cette couleur plus lumineuse. Toutefois pour ce diagramme « Erdgas » n'est pas important du tout.

Edward Tufte appelle les diagrammes comme celui qui précède « diagrammes de pacotille » (*chart junk*).

Voici quelques conseils qui pourraient vous aider à éviter de produire des diagrammes de pacotille.

— Ne pas utiliser de diagramme circulaire en 3D. C'est *mal.*

— Envisager d'utiliser une table au lieu d'un diagramme circulaire.

— Ne pas appliquer les couleurs au petit bonheur la chance ; les utiliser pour guider l'attention du lecteur et pour grouper les choses.

— Ne pas utiliser de motifs d'arrière-plan comme des hachures ou des diagonales au lieu de couleurs. Ces motifs perturbent. De tels motifs dans un diagramme instructif sont *mal.*

3.8. Attention et distraction

Prenez votre roman favori et jetez un œil sur une page typique. Vous verrez que la page est très uniforme. Rien n'est là pour perturber le lecteur dans sa lecture ; pas de gros titres, pas de texte en gras, pas de grande plage blanche. En fait, même quand l'auteur veut mettre quelque chose en évidence, on le fait avec des italiques. De tels caractères se fondent correctement dans le texte principal — de loin vous ne pourriez pas dire quelle page contient des italiques mais vous verriez immédiatement un seul mot en gras. Si les romans sont composés comme cela c'est pour suivre le paradigme : évitez ce qui perturbe.

Une bonne typographie (comme une bonne organisation) est quelque chose que l'on *ne remarque pas.* Le boulot de la typographie est de rendre la lecture du texte, c.-à-d. l'absorption des informations qu'il contient, aussi aisée que possible. Pour un roman, le lecteur absorbe le contenu en lisant le texte ligne à ligne comme s'il écoutait quelqu'un lui racontant l'histoire. Dans ces circonstances tout ce qui dans la page empêche le regard d'avancer rapidement et régulièrement d'une ligne à l'autre rendra le texte plus difficile à lire.

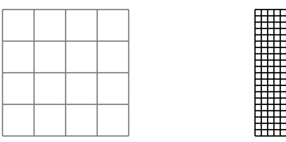
Maintenant, prenez un de vos magazines préférés ou un journal et regardez une page typique. Vous remarquez que beaucoup de choses « se passent » sur la page. Les polices sont utilisées à des tailles différentes et dans des arrangements différents, le texte est organisé en colonnes étroites, souvent entrelacées de photos. Si les magazines sont mis en pages de la sorte c'est à cause du paradigme : attirez l'attention.

On ne lit pas un magazine comme on lit un roman. Au lieu de lire un magazine ligne à ligne, on utilise les titres et les résumés pour savoir si l'on veut lire ou pas un certain article. Le boulot de la typographie est d'attirer notre attention d'abord sur ces résumés et ces titres. Une fois que l'on a décidé de lire un article, toutefois, on ne tolère plus d'être distrait, c'est pourquoi le texte principal de l'article est composé de la même manière que celui d'un roman.

Les deux principes « évitez ce qui perturbe » et « attirez l'attention » s'appliquent aussi aux graphiques. Lorsque l'on crée un graphique, on devrait éliminer tout ce qui va « distraire l'œil ». En même temps, on devrait essayer d'aider activement le lecteur « à travers le graphique » en utilisant des paramètres pour les polices, couleurs, épaisseurs de ligne qui mettent en relief les différentes parties.

Voici une liste non exhaustive des choses qui peuvent perturber un lecteur :

— Les contrastes importants seront enregistrés les premiers par l'œil. Par exemple, considérons les deux grilles suivantes :



Bien que la grille de gauche soit la première dans l'ordre normal de lecture, la droite a la plus de chance d'être vue la première : le contraste blanc sur noir est plus grand que celui du gris au blanc. De plus, il y a plus de « places » ajoutant à l'effet global de contraste dans la grille de droite.

Les choses comme des grilles et, plus généralement, toutes les lignes d'aide ne devraient pas empoigner l'attention du lecteur et donc devraient être tracées avec un contraste faible avec l'arrière plan. Par ailleurs, une grille avec un maillage lâche perturbe moins qu'une grille au maillage serré.

— Les lignes de pointillés créent de nombreux points où apparaît un contraste du noir au blanc. Les pointillés et les tirets peuvent être très perturbants et, aussi, devraient être généralement évités.

Il ne faut pas utiliser des motifs différents de pointillés pour distinguer des courbes. De cette façon on perd des points de données et l'œil n'est pas particulièrement bon à « grouper les choses en fonction d'un motif de pointillés ». L'œil est *bien* meilleur à grouper en fonction des couleurs.

— Les motifs de fond, qui remplissent une surface avec des diagonales, des horizontales ou des verticales ou même des points, sont presque toujours perturbants et, en général, n'ont aucun but véritable.

— Les images de fond et les dégradés perturbent et il est rare qu'ils ajoutent quelque chose d'important au graphique.

— De jolis petits *cliparts* peuvent détourner l'attention des données.

INDEX

◖ Till TANTAU
Institut für Theoretische Informatik
Universität zu Lübeck
tantau@users.sourceforge.net
http://sourceforge.net/projects/pgf/